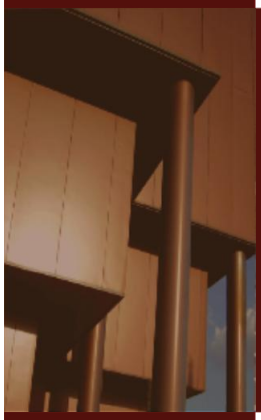
Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Generación de datasets multi-sensor para mapeado y localización sin GPS de aeronaves no tripuladas

Autor: María Polvillo Díaz
Tutor: José Ramiro Martínez de Dios



Dep. de Ingeniería de Sistemas y Automática
Grupo de Robótica, Visión y Control
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Generación de datasets multi-sensor para mapeado y localización sin GPS de aeronaves no tripuladas

Autor:
María Polvillo Díaz

Tutor:
José Ramiro Martínez de Dios
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Grupo de Robótica, Visión y Control
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Generación de datasets multi-sensor para mapeado y localización sin GPS de aeronaves no tripuladas

Autor: María Polvillo Díaz

Tutor: José Ramiro Martínez de Dios

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

Agradecimientos

A mi familia, a quien debo todo lo que tengo,
a mis compañeros del grupo Carmen, David, Luisma,
y en especial a Jenny, por ayudarme y enseñarme tanto desde el primer día
(y por dejarme un pedacito de su mesa),
y a Ramiro, por apostar por mí.

Resumen

Este trabajo nace como respuesta a la participación en el proyecto internacional AEROARMS cuyo objetivo es el desarrollo del primer sistema robótico aéreo del mundo con múltiples brazos articulados y capacidades avanzadas de manipulación para ser aplicadas en mantenimiento e inspecciones industriales. En concreto, el objetivo de este proyecto es la generación de *datasets* con sensores de distinta naturaleza que permitan posteriormente resolver con garantías el problema de mapeado y localización de dicho UAV sin necesidad de datos de GPS. Se exponen resultados de la aplicación de distintas técnicas de SLAM (del inglés, *Simultaneous Localization and Mapping*) y se realiza su evaluación, obteniendo conclusiones y realizando propuestas de mejora y posibles investigaciones futuras.

Abstract

This work arises from the participation in the international project AEROARMS which objective is the development of aerial robots with multiple arms for the first time in the world. The main challenge of the project is the application to inspection and maintenance in factories, specially in oil and gas plants. In particular, the objective of this work is the generation of datasets with different sensors that allow to solve the problem of mapping and localization of that unmanned robots without GPS. Results of different SLAM techniques are shown and they are evaluated to get conclusions that make it possible to suggest improvements and future research lines.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1. Antecedentes

Un vehículo aéreo no tripulado (del inglés *Unmaned Aerial Vehicle*, UAV) es una aeronave que consigue volar sin un piloto abordo. Puede ser teleoperado por un humano o volar autónomamente gracias al ensamblaje de autopilotos, sistemas de control de vuelo y módulos de procesamiento del movimiento del robot.

Aunque en su origen estas aeronaves eran utilizadas únicamente para servicios militares, su uso se ha ido extendiendo hacia otras aplicaciones llegando incluso al ámbito recreativo. Sin embargo, ha sido en el plano científico donde su desarrollo ha cobrado mayor importancia ya que la acción de estos UAV supone, en la mayoría de las ocasiones, la sustitución del hombre en situaciones peligrosas o la posibilidad de acceder con robots a lugares donde antes era una tarea imposible.

En la última década los UAVs han mejorado su autonomía tanto energéticamente como computacionalmente lo que ha permitido dotar a estos robots de capacidades sensoriales y de procesamiento, obteniéndose así un gran avance en tareas de seguimiento y localización. Existen ya aplicaciones en las que incluso varios UAVs realizan misiones de forma colectiva. Un ejemplo de esto es el estudio realizado en el proyecto multi-UAV COMETS [1] para la detección y localización de fuegos a través de cámaras visuales e infrarrojas. La cooperación de la información proporcionada por varios UAVs heterogéneos llega a ser muy importante en la disminución de la incertidumbre en la detección del fuego y en el aumento de la precisión en su localización.

Recientemente se ha creado un nuevo concepto de vehículo aéreo no tripulado. El estudio realizado por [2] propone leyes de control para UAVs equipados con un brazo robótico lo que, evidentemente, lo dota para realizar aplicaciones de manipulación aérea e incluso rescate de otros robots o UAVs.

Para que los UAVs operen de forma autónoma se requiere de una buena estimación de su posición en tiempo real en el entorno en que se mueve. En la mayoría de las ocasiones dicho entorno es desconocido por lo que se hace necesario realizar un mapa previo para una posterior localización. Existen técnicas de navegación que permiten al robot la generación de dicho mapa a la vez que se localiza dentro de él, lo que se conoce como SLAM, *Simultaneous Localization and Mapping*. El método de SLAM elegido en cada caso dependerá de factores como la resolución deseada del mapa o el conjunto de características que se reconocerán del entorno para realizar dicho mapa, conocidas como *features* [3].

Este trabajo nace como respuesta a uno de los objetivos marcados dentro del ámbito del proyecto internacional AEROARMS [4]. Está fundado por el proyecto Horizon 2020 para la Investigación y Desarrollo de la Comisión Europea y tiene como coordinador al profesor Aníbal Ollero [5]. Su principal meta es desarrollar el primer sistema robótico aéreo del mundo con múltiples brazos articulados y capacidades avanzadas de manipulación para ser aplicadas en mantenimiento e inspecciones industriales. Su logotipo se muestra en la figura 1.1.

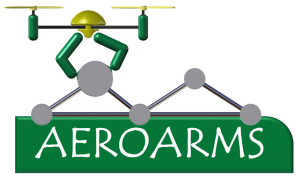


Figura 1.1: Logotipo del proyecto AEROARMS

AEROARMS continúa el trabajo llevado a cabo en el proyecto europeo ARCAS [6] en el que se desarrolló el primer robot aéreo con brazos de 6 y 7 grados de libertad con capacidades de percepción y planificación. Las imágenes 1.3a y 1.3b muestran los resultados obtenidos. AEROARMS incluye la demostración de los avances desarrollados en industrias que tienen grandes costes de mantenimiento, como son las de gas y gasóleo, y en el futuro su uso podría tener un gran impacto económico además de el desempeño de tareas que pueden ser peligrosas para los trabajadores.

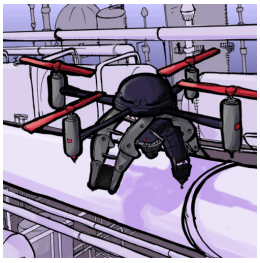


Figura 1.2: Representación de un UAV con varios brazos robóticos realizando operaciones de mantenimiento en un entorno industrial

1.2. Objetivos

El objetivo de este proyecto es la generación de sets de datos que recojan información del vuelo de un UAV que cuente con un lasér 3D, una cámara estéreo y una unidad de medición inercial abordo. El vehículo también dispondrá de un sensor de rango que hará de nodo monitor y que será el encargado de recibir datos de distancia de diversos nodos iguales que éste colocados en distintas posiciones del espacio en el que se realice el vuelo.

El objetivo a largo plazo es que la generación de estos datasets ofrezcan la posibilidad de emplear diferentes técnicas de mapeado y localización utilizando la información de cada uno de los sensores, entre las que

se incluirían técnicas de odometría y SLAM visual e incluso con los sensores de rango, *Range-Only* SLAM, como puede ser el algoritmo desarrollado en [7].

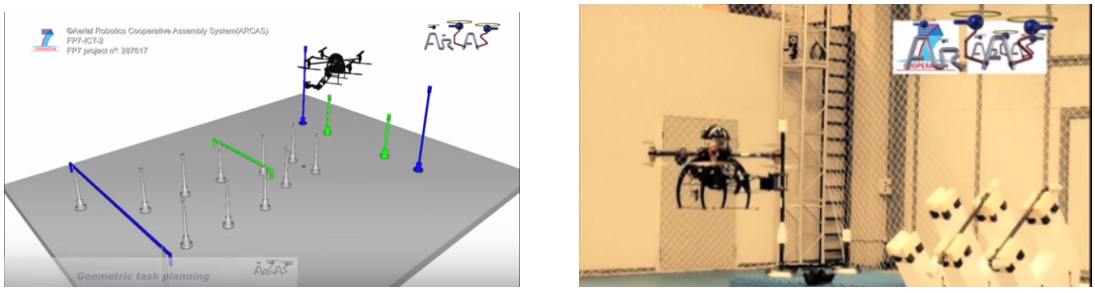
Como resultado se obtendrán varios mapas y la trayectoria seguida en cada uno de ellos. Aunque estos serán de distinto tipo ya que contendrán información diferente, tendrán en común tanto el movimiento del robot (los sensores están abordo del mismo UAV) como el entorno en el que se mueve. Adicionalmente, para aumentar la información obtenida de dicho entorno, se podrían adjuntar *markers* a los nodos y utilizar métodos de detección como el descrito en [8].

Hallando las características que tienen en común los mapas se podría hacer una superposición de la información que contiene cada uno de ellos, lo que disminuiría los errores tanto del mapa como de la trayectoria estimada, llegándose a obtener precisiones mucho más pequeñas relativamente que las que ya proporcionan los sensores utilizados. A modo de ejemplo, uno de los algoritmos que se podrían emplear para dicha superposición es el descrito en [9]. Con este algoritmo se puede hallar la matriz de transformación que mejor se ajusta a los ejes coordenados de los *features* que definen dos mapas diferentes mediante mínimos cuadrados.

1.3. Estructura del trabajo

En cuanto a la estructura seguida, en primer lugar se va a realizar un estudio del estado del arte de las técnicas de localización y mapeado simultáneos desarrolladas hasta la actualidad y se va a describir el sistema bajo el que se ha desarrollado la parte *software* de este proyecto.

Una vez hecho esto se procede a la explicación del diseño hardware del sistema, que conforma el primero de los tres bloques principales de este este trabajo. En este capítulo se presenta el láser 3D, la cámara estéreo, la unidad de medición inercial y los sensores de rango utilizados, explicando el principio de funcionamiento y puesta en marcha de cada uno de ellos así como sus especificaciones, destacando los medios requeridos para su uso y sus características principales.



(a) Simulación (b) Experimento real

Figura 1.3: Demostración de los resultados obtenidos en el proyecto europeo ARCAS

Capítulo 2

Estado del Arte

2.1. Introducción

SLAM (del inglés, *Simultaneous Localization and Mapping*) es una tarea esencial para la autonomía de un robot. Abarca el problema de la construcción de mapas de entornos desconocidos a priori por el robot al mismo tiempo que se localiza dentro de dicho mapa. Actualmente, el problema del SLAM está considerado resuelto cuando se usan sensores láser 2D o sonar en ambientes estáticos y pequeños. Sin embargo, las técnicas de SLAM basadas solamente en sensores visuales y en ambientes complejos o dinámicos todavía es un área activa en cuanto a investigación [10].

En este capítulo se va a realizar una presentación del estado del arte de los métodos de mapeado y localización existentes. Lo que se pretende es obtener una visión global y actualizada de las distintas técnicas SLAM que se han realizado hasta el día de hoy y los autores que están haciendo investigación en este ámbito, lo que permitirá realizar propuestas de futuras modificaciones e investigaciones. Posteriormente se va a hacer una introducción a ROS (del inglés, *Robot Operating System*), que no es más que el conjunto de librerías, herramientas y protocolos utilizado para el desarrollo software realizado en este proyecto.

2.2. SLAM

SLAM (del inglés, *Simultaneous Localization and Mapping*) surge de la necesidad de construcción de un mapa del entorno mientras se deduce la posición del robot dentro de ese mapa [11]. El vehículo debe tener capacidades sensoriales para detectar los distintos puntos característicos del entorno para obtener su posición respecto a ellos [12]. Los sensores comúnmente usados son de tipo láser, sonar y de visión, aunque también suelen complementarse con la información proporcionada por otras fuentes como por ejemplo el Sistema de Posicionamiento Global(GPS) [3].

Los sensores láser, como el utilizado en este proyecto, son muy precisos. Su principio de funcionamiento se basa en emitir un pulso en forma de rayo láser y medir el tiempo que tarda en reflejarse y volver hasta el dispositivo. Por otra parte, los sistemas sonar [1] son rápidos aunque dependen de sensores inerciales cuyos errores

[1] del inglés SONAR, acrónimo de Sound Navigation And Ranging

pueden producir efectos negativos en la estimación de la posición. Los sensores de visión están caracterizados por tener un gran rango y una gran resolución aunque el coste de computación requerido es relativamente alto [11].

En la mayoría de las ocasiones suele aplicarse SLAM para casos en que el vehículo es terrestre y principalmente para espacios en interiores, como son los estudios realizados en [13] [14] [15] [16]. Sin embargo, pocos son los estudios en los que se realiza simultáneamente el mapeado y la localización con vehículos aéreos [17] [18], como es el caso de este proyecto, y menos aún para aplicaciones desarrolladas en entornos sumergidos [19], tema que está despertando bastante interés.

Uno de los problemas fundamentales del SLAM es la asociación de datos. Detectar dos puntos característicos (conocidos como *features*) del mapa que se corresponden con el mismo punto en distintas posiciones y en distintos instantes de tiempo y reconocer que dichos puntos son el mismo. La resolución de este problema permite, por una parte, corresponder dos escenas sucesivas y, por otra parte, cerrar un bucle de la trayectoria realizada por el robot cuando éste vuelve a la posición de origen, problema conocido como *loop closing*. El uso de sensores visuales ofrece la posibilidad de extraer puntos singulares del entorno proporcionando información 2D y 3D por lo que se hace más robusta la selección de los *features* [20] [21]. Los avances conseguidos en el procesamiento de datos visuales permiten detectar puntos del mapa con características más complejas que las simples formas geométricas. Sin embargo, el uso de visión tiene ciertas limitaciones ya que requiere hacer suposiciones del entorno con objeto de simplificar el problema de detección [11]. Finalmente, una vez que se ha hecho el *loop closing*, se suelen aplicar técnicas, como por ejemplo [22], para corregir los desajustes provocados en el mapa debido a los errores en la estimación de la trayectoria.

En este proyecto se han utilizado dos tipos de SLAM visual diferentes, que permiten obtener resultados en una dimensión más que la obtenida con sensores sonar o láser. Actualmente, este tipo de técnicas de mapeado y localización están bastante activas en el ámbito de la investigación [10] ya que las técnicas de visión por computador empleadas son susceptibles de ser mejoradas, como por ejemplo la detección de *features*, los descriptores y el problema de *matching* y la asociación de datos, explicados en el párrafo anterior.

En la última década, ha habido una gran tendencia en la publicación de artículos en los que se emplean sensores visuales como la única fuente de información para resolver el problema del SLAM [23] [24]. La principal razón es que con estos dispositivos se puede extraer datos de rango del entorno así como de color, textura y apariencia, que también permiten dotar al robot de la capacidad de realizar tareas como detección y reconocimiento de lugares y personas. Sin embargo, los cambios de iluminación, lugares con poca textura y la poca resolución de la cámara, entre otros, pueden producir ciertos errores en los datos.

El primer trabajo de navegación visual realizado se basó en una cámara binocular estéreo [25]. Existen trabajos posteriores en los que se han usado múltiples cámaras con o sin solapamiento de sus visiones [26] [27], y cámaras omnidireccionales [28] con el objetivo de aumentar el rango de visión para reducir la acumulación de errores en la estimación de la posición. Recientemente, se han utilizado sensores RGB-D [29] [30] en el mapeado de interiores demostrando ser una alternativa prometedora para aplicaciones SLAM.

Como se ha comentado en el inicio de este capítulo, el desarrollo *software* del proyecto se ha realizado utilizando lo que en inglés se conoce como *Robot Operating System*. A continuación se procede a la introducción del mismo destacando las características con las que ha sido diseñado, la nomenclatura utilizada para denominar a sus componentes básicos, así como el método que ofrece para establecer relaciones entre los distintos ejes

de referencia de cada uno de los elementos del sistema robotizado con el que se esté trabajando.

2.3. ROS

ROS está definido como un conjunto de librerías, herramientas y protocolos que facilitan el desarrollo de aplicaciones para robots. Fue originado en el año 2007 con el nombre de *switchyard* por el Laboratorio de Inteligencia artificial bajo el respaldo del proyecto Stanford AI Robot (STAIR). En el año 2008, más de veinte instituciones, entre las que se encontraba el instituto Willow Garage, colaboraron en su desarrollo para hacerlo público bajo licencia BSD en el 2009.

2.3.1. Características de ROS

Las características que se pueden destacar de ROS son [31]:

- Comunicaciones par-par
- Basado en herramientas
- Multilenguaje
- Transparente
- De software libre

En cuanto a la primera de ellas, un sistema que se encuentre operando bajo ROS no es más que un conjunto de *hosts* diferentes conectados en una red con topología par-par (comúnmente conocida en inglés como *peer-to-peer*). Este tipo de topología requiera de mecanismos de búsqueda conocidos en ROS como *servicios* o *maestro*, descritos más adelante.

Una de las cosas que hacen que el entorno de trabajo de ROS no resulte monótono es que ROS está basado en pequeñas herramientas que pueden realizar tareas como navegar en el árbol de códigos fuente, visualizar las conexiones par-par que existen en un momento dado como muestra la figura , mostrar gráficamente datos de mensajes recibidos, etc.

Una de las cosas que hacen que el entorno de trabajo de ROS no resulte monótono son las pequeñas herramientas con las que cuenta. Estas pueden realizar diversas tareas como por ejemplo navegar en el árbol de códigos fuente, mostrar gráficamente datos de mensajes recibidos y visualizar las conexiones par-par que existen en un momento dado entre procesos, como se muestra la figura 2.1.

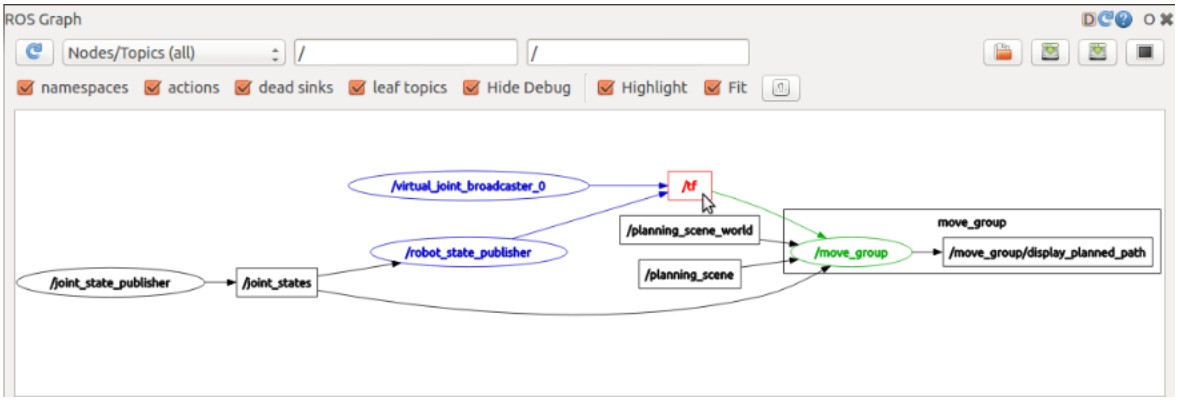


Figura 2.1: Herramienta de ROS que permite ver las conexiones entre los distintos procesos en ejecución

Siguiendo el orden de las características expuestas anteriormente, que ROS sea un sistema multilenguaje permite la flexibilidad en la programación en diversos entornos según las preferencias individuales de cada usuario. Los lenguajes de programación que soporta ROS son C++, Python, Octave y LISP.

ROS fomenta el uso de librerías estándar que no dependan de él. El objetivo de esto es posibilitar que los algoritmos y *drivers* desarrollados puedan ser utilizados en entornos fuera de ROS. Además, ROS reusa código de otros proyectos de software libre como pueden ser *drivers*, sistemas de navegación, algoritmos de visión de OpenCV [32] y algoritmos de planificación de OpenRAVE [33].

ROS

Open Source Robotics Foundation

Figura 2.2: ROS, una plataforma de software libre

Por último, tal y como se detalla en la figura 2.2, todo el código de ROS está disponible lo que supone una característica bastante atractiva. La licencia con la que se publican es BSD, que permite el desarrollo tanto de proyectos comerciales como no comerciales.

2.3.2. Nomenclatura de ROS

Una vez destacadas las principales características de la filosofía con la que se ha desarrollado ROS se va a proceder a explicar sus conceptos básicos, *nodos*, *mensajes*, *topics* y *servicios* [31].

ROS denomina a los procesos como *nodos*. Pueden ser ejecutados de forma paralela lo que hace que el diseño de ROS sea modular y escalable.

La comunicación entre dichos procesos se realiza mediante *mensajes* que pueden enviarse a través de dos canales diferentes, *servicios* y *topics*. Ambos son una simple cadena de caracteres. El primero de ellos sólo posibilita la conexión de dos nodos entre los que únicamente se envían mensajes de petición-respuesta (relación *servidor-cliente*). Los *topics* publicados por cualquier nodo (*publicador*), sin embargo, pueden tener más de un nodo subscrito (*subscriptor*) y todos ellos recibirán el mensaje que porta dicho *topic*. La imagen 2.3 recoge esto de forma esquemática.

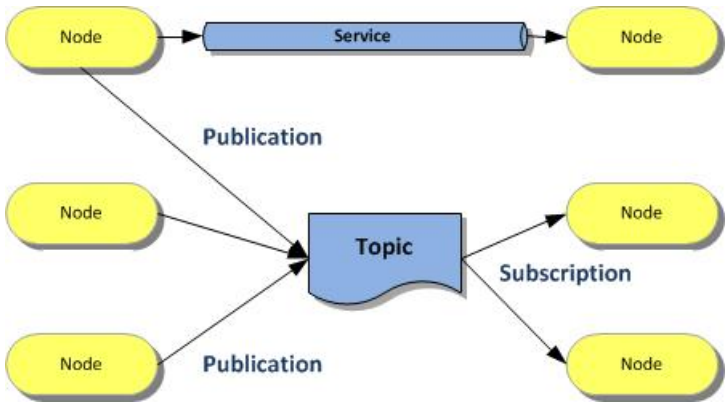


Figura 2.3: Diagrama de bloques de las posibles formas de comunicación entre nodos de ROS

La interoperatividad entre software y hardware desarrollado por personas o entidades diferentes facilitando su uso a ingenieros, investigadores e incluso personas aficionadas a la robótica, se realiza mediante contribuciones de proyectos bajo la forma de lo que en ROS se denominan *paquetes*. Los *paquetes* son simplemente directorios que contienen un fichero de extensión *xml* donde se describe y se establecen sus dependencias. Se pueden realizar directorios que contengan un árbol de paquetes, formando un *repositorio*.

Otra de las utilidades que ofrece ROS son los *bags*, lo que constituye el objetivo final de este proyecto. Son ficheros específicos del sistema donde se pueden guardar los *topics* que se estén publicando en ese momento para poder reproducirlos posteriormente, lo que ofrece la posibilidad de almacenar los datos de experimentos completos para poder trabajar sobre ellos después, analizarlos e incluso, al igual que los *paquetes*, compartirlos.

Finalmente, una vez presentados los conceptos básicos de ROS, se va a proceder a explicar el sistema de transformaciones desarrollado en ROS.

2.3.3. Transformaciones en ROS

Tal y como se describe en [31], los sistemas robotizados necesitan de relaciones espaciales en multitud de casos como puede ser, por ejemplo, para la localización de un robot móvil respecto a un sistema de referencia fijo y relaciones entre los sistemas de coordenadas de varios sensores montados sobre un robot manipulador.

Con el fin de simplificar el tratamiento de coordenadas espaciales se ha desarrollado en ROS un método de transformación que ha sido denominado como *tf*. Éste realiza los cálculos necesarios para relacionar todos los ejes de referencias existentes en cada uno de los componentes del sistema con el que se esté trabajando.

A modo de ejemplo, *tf* puede ser utilizado para generar nube de puntos en un sistema de coordenadas estático, "map", de las medidas proporcionadas por un sensor láser que esté abordo de un robot móvil. Este tipo de operaciones pueden ser tediosas de realizar directamente por el usuario pero no lo son tanto en ROS gracias a la implementación de *tf*.

2.4. Conclusión

A modo de conclusión, del análisis realizado en este capítulo se puede deducir que la mayoría de las técnicas de SLAM se han aplicado con robots terrestres [13] [15], existiendo pocos estudios en los que se hayan usado UAVs [17] [18] [34]. Además, son pocos los trabajos en los que se haya aplicado SLAM en situaciones en las que el robot esté sumergido bajo agua y menos aplicando sensores visuales. Existen diversas propuestas de SLAM visual pero la mayoría de ellas están destinadas a robots terrestres [13] [35] [36] [37]. Con los *datasets* obtenidos en este proyecto se podrán aplicar técnicas de SLAM utilizando sensores láser 3D [38], SLAM visual [39] y algoritmos de RO SLAM [2] [7], y se hará uso de los recursos que proporciona ROS para su implementación.

[2] *Range Only SLAM*

Capítulo 3

Diseño hardware del sistema

3.1. Introducción

Para hacer que cualquier robot tenga cierta autonomía es necesario equiparlo con dispositivos que lo doten de capacidades sensoriales. Una vez conocida la tarea que éste debe realizar y los requisitos que deben cumplir los resultados obtenidos, lo primero de todo es determinar qué tipo de sensores utilizar. Se deberá tener conocimiento de las nociones de cada uno ellos para saber si se adaptan al entorno en el que se va a desenvolver el robot e incluso si se adaptan al propio robot en sí (se deberá tener en cuenta el tamaño y el peso si debe ir abordo, por ejemplo). Una vez establecido el tipo de sensor o sensores, la elección de uno u otro dentro de la misma familia vendrá determinada por ciertas características, como pueden ser su alcanzabilidad, la precisión y el error en la medida, críticas en la calidad de los resultados obtenidos.

En este capítulo se va a hacer una breve introducción de los sensores seleccionados para la tarea que se plantea en este proyecto, se va a detallar el principio de funcionamiento y puesta en marcha de cada uno de ellos así como sus especificaciones y características técnicas. Además se va a describir el ordenador de abordo que permitirá la implementación de los sensores en el vuelo del UAV.

3.2. Velodyne LIDAR

Velodyne es un sensor láser 3D de alta precisión de tipo LIDAR (Light Detecting And Ranging) con una cabeza giratoria que contiene un determinado número de láseres semiconductores [40]. Cada uno de los láseres tiene su propio detector.

Esta tecnología fue originalmente diseñada para competir en el *Defense Advanced Research Project Agency's* (DARPA) "Grand Challenge", una competición patrocinada por el Departamento de Defensa de los Estados Unidos para estimular la innovación en el campo de los vehículos terrestres no tripulados. El objetivo era conseguir un vehículo que pudiera navegar autónomamente a través del desierto. Actualmente los sensores LIDAR de alta precisión han sido extendidos a otras muchas industrias.

En concreto para este proyecto se ha utilizado el modelo HDL-32E, mostrado en la figura 3.1, que dispone de 32 láseres con un campo vertical de visión de 40º. Tiene una precisión de $\pm$ 2 cm y un rango de 80-100 m.

Figura 3.1: Velodine LiDAR HDL-32E

3.2.1. Principio de funcionamiento y puesta en marcha

El sensor HDL-32E crea imágenes 3D de 360º gracias al giro de su cabeza donde dispone de 32 pares láser-detector. El sensor proporciona una nube de puntos formada por la medida recogida por los miles de emisiones de pulsos de luz por segundo de cada uno de los láseres de que dispone. Esta nube de puntos deberá ser procesadas por el ordenador al que esté conectado el sensor para interpretar el entorno, como se muestra en la figura 3.2.

De la misma forma en que es representado en la figura 3.3, cada sensor láser mide el tiempo de vuelo o tiempo que tarda en volver el pulso de luz que emite para proporcionar la distancia a la que se encuentran los objetos que son detectados. Conociendo que la velocidad a la que viaja dicho pulso es aproximadamente la velocidad de la luz (300 millones de metros por segundo), la distancia puede ser conocida.

Otra característica a tener en cuenta de la medida es que la nube de puntos está compuesta de bandas claramente identificables, correspondientes a los diferentes canales (láseres). Cada uno de ellos se encuentra inclinado respecto de la horizontal un determinado ángulo con el objetivo de conseguir un mayor campo de visión vertical, de 41.3º en el caso del Velodyne HDL-32E. De este modo, cuanta mayor sea la distancia a la que se encuentran los objetos, mayor se hará la distancia entre las bandas de la nube de puntos y por consiguiente menor será la precisión obtenida por el sensor, concluyendo que ésta será mayor para objetos que estén relativamente cerca. Concretamente, para el modelo en cuestión, las recepciones serán válidas hasta algo más de 70 metros.

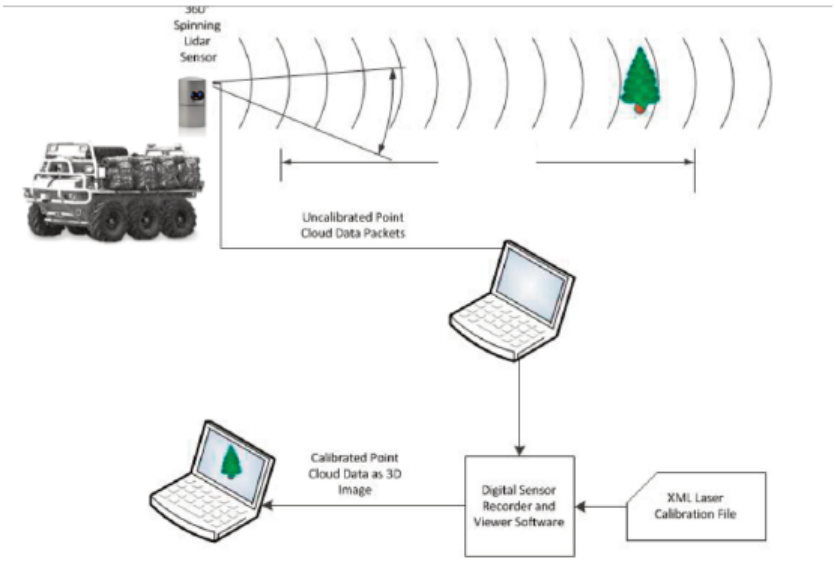


Figura 3.2: Visión general del proceso de imagen del LiDAR HDL-32E

En cuanto a la puesta en marcha del sistema, una vez que éste está correctamente montado y alimentado, lo siguiente es conectarlo mediante un conector Ethernet a un puerto Ethernet RJ45 de una computadora. Sin necesidad de configuración o calibración el sistema empezará a generar datos emitidos en dos paquetes UDP separados. Cada HDL-32E tiene su propia dirección MAC basada en su número de serie. En cambio, para todos ellos la dirección IP es la 192.168.3.255.

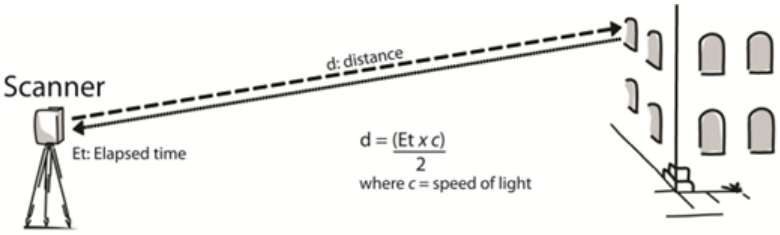


Figura 3.3: Láser como un "bastón"para estimar medidas

Una vez que se ha establecido la comunicación mediante el puerto Ethernet entre la computadora y el dispositivo, se debe proceder a la calibración del mismo. De forma general se suele crear una tabla de calibración a partir del archivo db.xml (proporcionado por el CD incluido con el HDL-32E) que deberá ser almacenada en el software encargado de procesar la nube de puntos. Para cada uno de los láseres se aplica una corrección en el ángulo vertical que, vistos desde la parte trasera del sensor, tendrá un valor positivo para aquellos láseres apuntando hacia arriba de la horizontal y negativos para aquellos que apuntan hacia abajo. De esta forma se puede calcular la posición de cada punto en el espacio 3D.

Podría ser interesante coordinar el sensor con otros sensores láser. El HDL-32E tiene la capacidad de sincronizar sus datos con tiempos de precisión de GPS. En el caso de utilizar el GPS proporcionado con el HDL-32E (Garmin GPS-18LV GPS) éste puede comunicarse con el sensor a través de la caja de interfaz de usuario. En el caso de utilizar un GPS distinto, el usuario debe configurarlo para que emita los datos necesarios secuencialmente en el tiempo requerido.

3.2.2. Especificaciones

La información técnica del Velodyne HDL-32E es la mostrada en la tabla 3.1.

Cuadro 3.1: Especificaciones Velodyne LiDAR

Láser	- Clase 1 - seguro para el ojo - 903 nm longitud de onda - Medida del tiempo de vuelo - Rango de medida de 1m a 70m - Máxima Altura de Operación de 2000m
Sensor	- 32 pares láser/detector - Campo de visión vertical de +10.67 a -30.67º - Campo de visión horizontal de 360º - 10 Hz de *frames* por segundo - Temperatura de operación de -10 ºC a +60ºC - Temperatura de almacenamiento soportada de -40 ºC a 105 ºC - Precisión de menos de 2cm - Resolución angular vertical de aproximadamente 1.33º - Resolución angular horizontal de aproximadamente 0.16º a 600 rpm
Mecánica / Eléctrica / Operacional	- Alimentación: 12V y 2 Amps - Voltaje de operación: 9-32 VDC - Peso: $<$ 2 kg - Vibración: 5Hz a 2000 Hz, 3 Grms - Protección del medio ambiente: IP67/Tipo 4

Salida	- Aproximadamente 700000 puntos por segundo
	- Conexión Ethernet de 100 Mbps
	- Paquetes UDP:
	  - distancia
	  - rotación
	- Corrección en orientación: acelerómetros MEMS internos y giróscopos para corrección en 6 ejes de movimiento
	- Tiempo de sincronización de GPS

Dimensiones (Altura/Diámetro)	- 5.68"x 3.36"[149.86 mm x 85.3 mm]
Peso del sensor	- 2.9 lbs [1.3 kg]
Peso de envío (aprox.)	- 14.35 lbs lbs [6.5 kg]
Longitud de onda	- 903 nm. Rango min/máx es de 896/910 nm
Duración de Pulso/Velocidad de Repetición	- 6 ns (duración) - 1.44 us*32 láseres resulta un período de 46.1 us = 21.7 kHz (repetición)
Potencia Máxima/Energía:	- 31.4 W(0.19 uJ)

La unidad está diseñada para resistir las vibraciones de automoción estándar, que tienen las características

descritas en la tabla anterior (500 m/s^2, 11 mseg de duración de choque y vibración de 3 Grms de 5 Hz a 2000 Hz).

Caja de interfaz

EL cable del sensor termina en una caja de interfaz (figura 3.4) que facilita la conexión Ethernet, la alimentación y la entrada de señal GPS.

(a) Vista superior (b) Vista frontal

Figura 3.4: Imagen de la caja de interfaz del dispositivo

Respecto a las características eléctricas, un valor lógico de 1 se corresponde con un voltaje que debe estar entre 3 V y 15 V. Un 0 lógico se corresponde con un voltaje menor de 1.2 V. La unidad GPS/INS debe ser capaz de proporcionar una corriente de al menos 2mA en el valor lógico 1. La polaridad del mensaje NMEA es como el mostrado en la figura siguiente.

Descripción de la señal del cable de interfaz

La señal de los cables de la caja de interfaz están recogidos en la tabla 3.2.

Cuadro 3.2: Señal del cable de interfaz

Cable	**Señal**	**Entrada / Salida**	**Especificaciones**
Negro	Tierra	Entrada	Tierra del sistema
Rojo	Alimentación	Entrada	9 - 15 VDC / 12 W
Amarillo	Pulso de sincronización GPS	Entrada	TTL
Blanco	Receptor GPS serie	Entrada	TTL
Naranja Claro	Ethernet TX+	Salida	Diferencial
Naranja	Ethernet TX-	Salida	Diferencial
Azul Claro	Ethernet RX+	Entrada	Diferencial
Azul	Ethernet RX-	Entrada	Diferencial

3.3. Cámara estéreo ZED

Una cámara estereoscópica es una cámara capaz de crear imágenes en 3 dimensiones. Intenta imitar el comportamiento de la visión humana utilizando dos cámaras separadas una cierta distancia que captan la fotografía en el mismo instante.

En concreto para este proyecto se ha utilizado la cámara ZED, de StereoLabs [41], cuya vista frontal puede verse en la figura 3.5. Es una cámara de profundidad basada en visión estereoscópica pasiva. Proporciona como salida vídeo de alta resolución que contiene flujos de vídeo de derecha e izquierda sincronizados y crea mapas de profundidad en tiempo real usando la unidad de procesamiento gráfica (GPU, *graphics processing unit*) de la máquina a la que esté conectada.

Figura 3.5: Vista frontal de la cámara estereoscópica ZED

3.3.1. Principio de funcionamiento y puesta en marcha

El principio de funcionamiento básico de la cámara ZED es el de cualquier cámara estéreo. Al igual que la visión humana, capturando de forma sincronizada dos imágenes con una determinada separación y procesándolas posteriormente se pueden conseguir imágenes con cierto efecto de relieve. Debido a la separación entre lentes estas cámaras no suelen ser precisas en profundidad para distancias relativamente cortas aunque sí que pueden llegar a detectar objetos hasta distancias de 50 cm, para la cámara ZED.

Para ponerla en funcionamiento es necesario tener una unidad de procesamiento gráfico potente en el dispositivo al que se vaya a conectar para poder obtener mapas de profundidad en tiempo real. Esta función es realizada por el software ZED SDK (*Kit* de Desarrollo de Software ZED) para el que hace falta que el sistema cumpla ciertos requisitos básicos:

- Procesador Dual Core de 2.3 GHz o más
- 4 GB de RAM o más
- GPU con NVIDIA con Capacidades Computacionales > 2.0
- CUDA 6.5
- Puerto USB 3.0
- Windows 7, Windows 8, Windows 8.1 (64 bits), Ubuntu 14.04, L4T21.3/4

Además si la resolución a la que se van a grabar datos es alta, por ejemplo 3840x1080 @30fps, es recomendable utilizar un disco duro que pueda alcanzar velocidades de 250 MB/s y que tenga al menos 256 GB de capacidad.

Si no se dispone de una GPU que sea NVIDIA, como por ejemplo Intel HD Graphics, se podrá utilizar la cámara para grabar vídeos 3D pero no se podrá visualizar profundidad.

La cámara está calibrada desde fábrica por lo que el usuario no tendrá que calibrarla cada vez que la ponga en funcionamiento. Cada unidad tiene sus propios datos de calibración englobados en un fichero. Este fichero es descargado automáticamente *online* desde la base de datos de ZED durante la puesta en marcha del software ZED SDK. Si durante la instalación de la cámara no se tiene acceso a Internet, ésta producirá datos pero estará descalibrada. Es posible obtener manualmente el fichero específico de la cámara, para ello:

1. El ordenador o dispositivo al que vaya a conectarse la cámara debe tener acceso a Internet
2. Conectar la cámara a un puerto USB 3.0
3. En la aplicación software *'ZED Settings App'*, hacer click en *'Factory Reset'* y luego en *'Save Configuration'*. Estas dos pestañas pueden verse en la figura 3.6 .

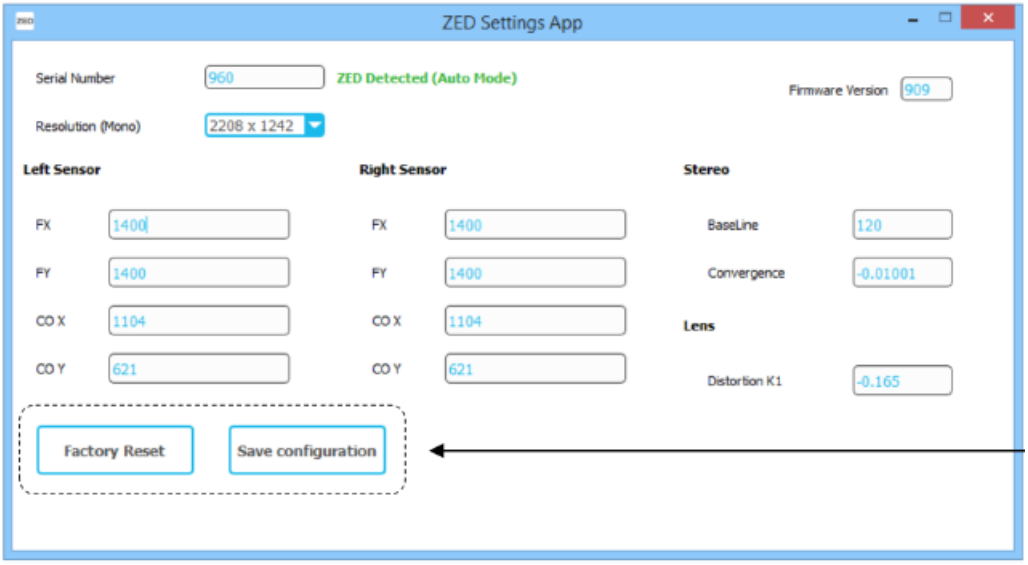


Figura 3.6: Imagen del software para la calibración de la cámara ZED

3.3.2. Especificaciones

La cámara ZED tiene las características técnicas englobadas en la tabla 3.3.

Cuadro 3.3: Especificaciones técnicas cámara estéreo ZED

Cámara	
Resolución de salida	2x(2208x1242)@15fps (2.2K) 2x(1920x1808)@30fps (1080p) 2x(1280x720)@60fps (720p) 2x(640x480) @100fps(VGA)
Formato de salida	YUV 4:2:2
Campo de visión	Máximo 110º (D)
Rango en profundidad	de 1 m a 15 m (3.5 a 49 ft)
Interfaz	USB 3.8 - cable integrado de 1.5 m
Campo de visión	Máximo 110º (D)
Electrónica	
Tipo de sensor	1/2.7"
Tamaño de matriz activa	4M píxeles por sensor
Tipo de sensor	1/2.7"
Distancia focal	2.8 mm (0.11") - f/2.0
Obturador	Obturador de rodamiento electrónico sincronizado
Tamaño del píxel	2 μm
General	
Dimensiones	175x30x33 mm (6.89x1.18x1.3")
Peso	159 g - 0.35 lb
Potencia	380mA / 5V alimentado por USB
Temperatura de operación	0ºC a 45 ºC (32 ºF a 113 ºF)

La resolución de la cámara puede cambiarse entre las cuatro posibles con la aplicación ZED Explorer, que también permite configurar otros parámetros como el tiempo de exposición, las imágenes por segundo, el balance de color y la ganancia.

3.4. Unidad de medición inercial (IMU)

Una unidad de medición inercial o IMU (*Inertial Measurement Unit*) es un sistema autosuficiente que mide movimiento linear y angular usando un conjunto de acelerómetros y giróscopos. Informa de la orientación, de la velocidad y de las fuerzas gravitatorias de un aparato.

Figura 3.7: Sistema MTi-G, de Xsens

En concreto la IMU que se ha usado en este proyecto ha sido el modelo MTi-G, mostrada en la figura 3.7. Además de acelerómetros y giróscopos, dispone de magnetómetro y barómetro por lo que procesa medidas de los ángulos de *Roll*, *Pitch* y *Yaw*, aceleraciones lineales en 3D, velocidad angular en 3D, campo magnético terrestre en 3D y presión atmosférica. Excepcionalmente entre los productos que ofrece Xsens, este dispositivo además cuenta con un sistema GNSS[1] (siglas en inglés de Sistema Global de Navegación) añadido que lo capacita para poder proporcionar valores de posición y orientación en 3D. Internamente, el procesador del sensor ejecuta en tiempo real un Filtro de Kalman (referido como XKF, del inglés Xsens Kalman Filter [42]), que calcula estimaciones de posición 3D y velocidad 3D.

3.4.1. Principio de funcionamiento y puesta en marcha

La interacción con el dispositivo se puede realizar utilizando comunicación serie RS-232/422/485 o USB caracterizada por estar basada en mensajes, lo que permite cambiar la configuración de la unidad.

La posición y orientación del MTi-G son estimadas utilizando un Filtro de Kalman Extendido y son obtenidas integrando a lo largo del tiempo los datos proporcionados por los sensores de inercia. Este proceso es conocido en inglés como *"dead reckoning"* o navegación por estimación. Dada una estimación de la orientación en el instante anterior, la velocidad angular proporcionada por los giróscopos es integrada para obtener una nueva estimación de la orientación. Igualmente, para estimar la posición actual se utiliza la previamente estimada y la integración doble de los datos dados por los acelerómetros. Debido a que la obtención de medidas se realiza de forma indirecta, si este método es utilizado durante largos períodos de tiempo se pueden llegar a producir grandes errores.

El receptor GPS complementa al sistema inercial proporcionando datos de posición y velocidad. De esta forma los errores que se producen al utilizar "*dead reckoning*"son corregidos, también en orientación. Se debe tener en cuenta que los datos recibidos del GPS son relativos a la posición y velocidad de la antena y no del MTi-G en sí, por lo que es necesario conocer su origen de coordenadas respecto del de la antena.

Las medidas del GPS son muy precisas en el plano horizontal pero lo son menos en el plano vertical. Para aumentar la precisión en altura el Filtro de Kalman hace estimaciones utilizando también las medidas de pre-

[1] GNSS son un conjunto de satélites que transmiten señales para proporcionar posición y localización en cualquier parte del planeta. Engloba a todos los sistemas de navegación como por ejemplo GPS, GLONASS y Galileo. Concretamente para el modelo MTi-G, el sistema utilizado es GPS

sión estática dada por un barómetro.

Para la puesta en marcha del dispositivo es necesario elegir dónde instalarlo en el objeto que se desea estabilizar, monitorizar o controlar. Por ejemplo, si se está utilizando la IMU para medir la dinámica de un vehículo en movimiento, la mejor posición sería aquella donde se produjeran menores aceleraciones transitorias que suele ser en el centro de gravedad del objeto o cerca de él.

Otro factor a tener en cuenta son las vibraciones ya que éstas son medidas directamente por los acelerómetros. Si la magnitud de las vibraciones o su frecuencia son mayores que el rango de medida o el ancho de banda de los acelerómetros, respectivamente, sus lecturas no serán válidas y podrían serlo también las de los giróscopos.

Si el sensor está cerca o montado sobre un material ferromagnético o en lugares donde se produzcan grandes intensidades entonces podrían producirse campos magnéticos que distorsionaran las medidas del magnetómetro y, por consiguiente, la medida del ángulo de *Yaw*.

No menos importante es la calibración de los sensores que componen la unidad para conseguir medidas precisas. Cada uno de ellos viene ya calibrado de fábrica. Los parámetros obtenidos dependen de cada sensor y se pueden consultar en el Certificado y Prueba de Calibración que viene con el kit del dispositivo.

3.4.2. Especificaciones

En las tablas 3.4 y 3.5 se muestran las características técnicas de la velocidad angular, aceleración, campo magnético, temperatura y presión atmosférica.

Cuadro 3.4: Especificaciones en velocidad angular y aceleración

		Velocidad Angular	**Aceleración**
Unidad		[grados/s]	[m/s^2]
Dimensiones		3 ejes	3 ejes
Escala máxima	[unidades]	$\pm$ 300	$\pm$ 50
Linealidad	[% of Escala máxima]	0.1	0.2
Tendencia a estabilidad	[unidades 1σ]	1	0.02
Estabilidad del Factor de Escala	[% 1σ]	-	0.03
Densidad de ruido	[unidades /$\sqrt{}$Hz]	0.05	0.002
Error de alineación	[grados]	0.1	0.1
Ancho de banda	[Hz]	40	30
Resolución A/D	[bits]	16	16

Cuadro 3.5: Especificaciones en campo magnético, temperatura y presión atmosférica

		Campo Magnético	**Temperatura**	**Presión Estática**
Unidad		[mGauss]	[^{o}C]	[Pa]
Dimensiones		3 ejes	-	-
Escala máxima	[unidades]	$\pm$ 750	-55 a +125	30 a 120 10^3
Linealidad	[% of Escala máxima]	0.2	$<$1	0.5
Tendencia a estabilidad	[unidades 1σ]	0.1	0.5	100/año
Estabilidad Factor de Escala	[% 1σ]	0.5	-	-
Densidad de ruido	[unidades /$\sqrt{}$Hz]	0.5(1σ)	-	$4^{(2)}$
Error de alineación	[grados]	0.1	-	-
Ancho de banda	[Hz]	10	-	-
Resolución A/D	[bits]	16	12	9

Especificaciones de funcionamiento

A continuación se describen las especificaciones técnicas del receptor GPS (tabla 3.6) y de la posición y orientación estimadas por el Filtro de Kalman implementado por Xsens (tabla 3.7).

[2]equivalente aproximadamente a 0.3m/$\sqrt{}$Hz

Cuadro 3.6: Especificaciones del receptor GPS

Tipo de Receptor	50 canales, GPS/QZSS L1, código C/A GALILEO OpenService L1
Frecuencia de actualización	4 Hz
Frecuencia de actualización en Posición/-Velocidad	120 Hz
Precisión en Posición SPS (3)	2.5 m CEP
SBAS (4)	2.0 m CEP (5)
Tiempo de arranque en frío	29 s
Readquisición	$<$ 1s
Sensibilidad de Seguimiento	-160 dBm
Precisión de tiempo	30 ns RMS
Límites Operacionales	
Altura Máxima	18 km
Velocidad Máxima	600 m/s (2160 km/h)
Dinamismo máximo del GPS	4 g

Cuadro 3.7: Posición y orientación del XKF-6G

Rango Dinámico	
Pitch	$\pm$ 90º
Roll	$\pm$ 180º
Yaw	$\pm$ 180º (0...360º)
Resolución Angular	0.05 º
Precisión Estática	
Roll/Pitch	$<$ 0.5 º
Yaw (6)	$<$ 1 º
Precisión Dinámica (7)	
Roll/Pitch	1 º RMS
Yaw (8)	2 º RMS
Máxima frecuencia de actualización Autonómica	120 Hz
PC/ datos sin procesar	512 Hz

Estos datos están sujetos a ciertas condiciones que pueden ser consultadas más detalladamente en la Documentación Técnica del MTi-G.

Sistemas de coordenadas

Los ejes de coordenadas fijos al sensor forman un sistema Cartesiano orientado según la mano derecha. La placa base de aluminio se coloca cuidadosamente durante la calibración en fábrica para que esté alineada con el sistema de referencia del dispositivo, teniendo una desviación máxima en orientación de 0.1 grados y una no ortogonalidad menor a 0.1 grados. Los datos de velocidad de giro, aceleración y campo magnético de salida están referido a este sistema de coordenadas.

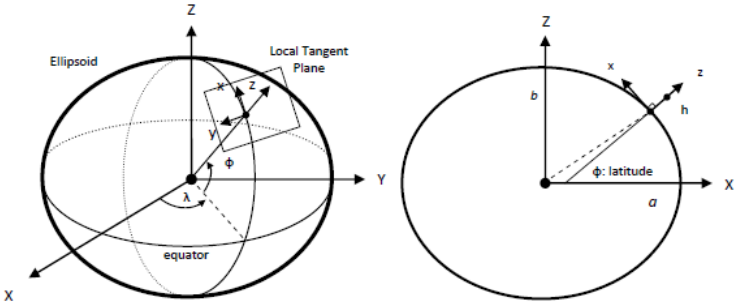


Figura 3.8: Definición del Plano Tangente Local Euclídeo por defecto, que es North West Up

Es posible rotar los ejes de referencia del MTi-G para que coincidan con los del objeto en el que está montado.

La unidad calcula la orientación del dispositivo tomando como referencia una linealización local del sistema de coordenadas Elipsoidal Terrestre (Longitud, Latitud y Altitud (LLA), según WGS84[9]) definido como en la figura 3.8 de forma que esquemáticamente ambos sistemas estarían como en la figura 3.9.

[3]GPS proporciona dos tipos de servicios de posicionamiento. PPS, del inglés Precise Positioning Service, y SPS, Standard Positioning Service. Este último está basado en el código C/A (Coarse/Adquisition) que está modulado sólo en la banda de frecuencia L1 transmitida por los satélites.

[4]Del inglés Satellite Based Augmentation System (Sistema de Aumentación Basado en Satélites) es un sistema de corrección de las señales que los Sistemas Globales de Navegación por Satelite (GNSS) transmiten al receptor del usuario. Mejoran su posicionamiento horizontal y vertical y dan información acerca de las señales.

[5]Depende de la precisión del servicio SBAS (son soportados WAAS, EGNOS, MSAS, GAGAN)

[6]Depende del escenario donde se esté utilizando. En el caso del campo magnético terrestre, debe ser homogéneo

[7]Bajo condiciones correctas del escenario configurado para el XKF, un filtro estabilizado y buena señal GPS.

[8]Depende del perfil utilizado para el XKF. En caso del campo magnético terrestre, debe ser homogéneo

[9]El WGS84 (del inglés, World Geodetic System 84) es un sistema de coordenadas geográficas mundial que permite localizar cualquier punto de la Tierra por medio de tres unidades dadas, sin necesidad de ninguna referencia. Consiste en un patrón matemático de tres dimensiones que representa el planeta por medio de una elipsoide. Tiene aproximadamente un error menor a 2 cm, motivo por el que es utilizado por el sistema GPS.

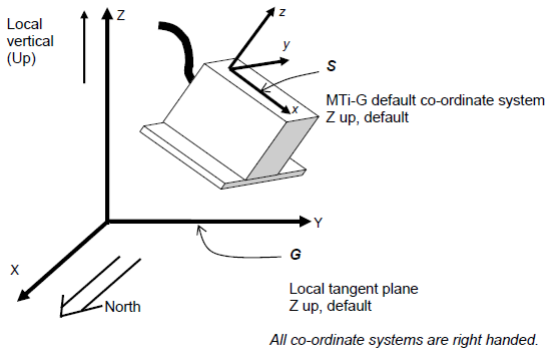


Figura 3.9: Orientación del sistema de coordenadas del MTi-G respecto del sistema de referencia

Tiempo de comunicación

Conocer con exactitud los retrasos o latencias en un sistema desde que se produce un evento físico hasta que el dispositivo consigue obtener una medida digital puede ser muy importante en muchas aplicaciones. En este caso, dicho retraso está determinado por dos factores que son, por una parte, el procesamiento de señal (tiempo de adquisición y cálculo de datos y generación de mensajes) y, por otra parte, la transmisión serie de los mensajes. Un esquema podría ser el de la figura 3.10.

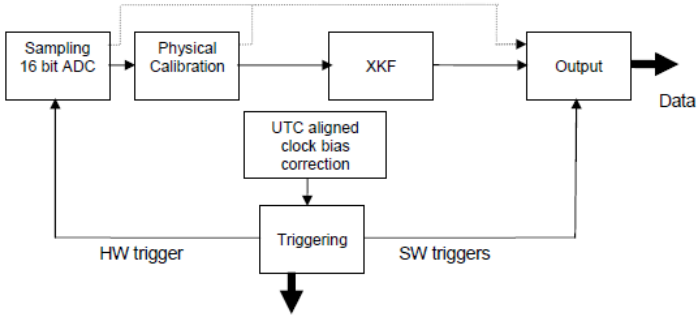


Figura 3.10: Procesos realizados por la unidad desde que se produce un evento físico hasta generar una salida digital

El tiempo de adquisición y cálculo es dependiente del escenario y del modo de salida. Además, el tiempo de computación no será constante porque el Filtro de Kalman realizará cálculos diferentes dependiendo de los datos disponibles de los sensores.

El tiempo de transmisión serie de datos puede obtenerse de la siguiente expresión si los bytes del mensaje y la velocidad de transmisión son conocidos:

$$Tiempo\ de\ transmisión = \frac{(bytes\ totales\ en\ el\ mensaje) * 10(bits/byte)}{velocidad\ de\ transmisión(bits/s)}$$

La configuración de la tabla 3.8 es la configuración por defecto de la conexión serie:

Cuadro 3.8: Configuración por defecto de la comunicación serie

Ajuste	**Valor por Defecto**
Bit/segundo (bps)	115200
Bits de los datos	8
Paridad	ninguna
Bits de parada	1
Control de flujo	ninguno

La velocidad de transmisión (bps) puede ser cambiada por el usuario. Como máximo puede ser 921600 bps y como mínimo 9600 bps.

Especificaciones físicas

Por último, las especificaciones físicas del MTi-G son las de la tabla 3.9:

Cuadro 3.9: Especificaciones físicas

Interfaz de Comunicación	Digital Serie (RS-232)
Interfaces Adicionales	
Voltage de Operación	5 - 30 V
Consumo de Potencia	Típica : 610 mW Máxima (sin señal GPS): < 910 mW
Rango de temperatura de operación	-20°C a 60°C
Rango específico de temperatura de funcionamiento	0°C a 55°C
Dimensiones	58 x 58 x 33 mm (W x L x H)
Peso	68 g

3.5. Kit de desarrollo nanoPAN 5375 de Nanotron Technologies

Las placas de este *kit* de desarrollo funcionan como sensores de rango y son generalmente usadas para el desarrollo de aplicaciones de localización inalámbrica. La alimentación y la transmisión de datos comparten el

mismo puerto USB y tienen una alta alcanzabilidad gracias a que su salida es de alta potencia, de +20 dBm [43].

A nivel modular, como puede verse en la figura 3.11, las dos partes de las que consta el *kit* son el *driver* nanoLOC nTRX, que proporciona una API[10] que funciona como interfaz para el módulo transmisor RF nanoPAN 5375, y un microcontrolador ARmeda 1284P y una JTAG para *debuguear* aplicaciones.

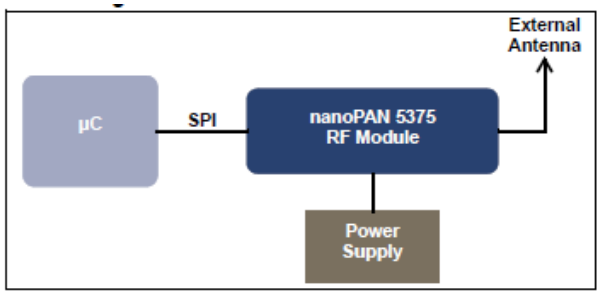


Figura 3.11: Diagrama estructural del módulo nanoPAN

3.5.1. Especificaciones de la placa del kit de desarrollo

Las placas nanoPAN DK como la mostrada en la figura 3.12, integran módulos RF nanoPAN 5375, un microcontrolador ATmega1284P, circuitería de carga, sensor de temperatura, adaptador JTAG, conectores USB, *socket* de expansión y botones y LEDs programables. Las antenas que utilizan son omnidireccionales de 2.4 GHz.

Figura 3.12: *kit* de desarrollo del módulo nanoPAN

En la tabla 3.10 se muestran las características generales de las placas del *kit*.

[10]Del inglés, Application Programming Interface, es un conjunto de funciones y subrutinas que ofrece cierta biblioteca para ser utilizada por otro software

Cuadro 3.10: Características de la placa del *kit* de desarrollo nanoPAN 5375

Módulo RF	Módulo RF nanoPAN 5375
Técnica de Modulación	propagación de espectro Chirp[11]
Banda de operación	Banda ISM de 2.4 GHz
Rango de precisión	2m en interiores / 1m en exteriores
Voltaje de alimentación	5V por USB
Potencia de salida (programable)	de -15 dBm a + 20dBm
Tasa de datos	de 250 kbps a 1 Mbps
Sensibilidad de recepción (FEC [12])	hasta -97 dBm
Corriente consumida	500 mA

3.5.2. Especificaciones del módulo RF nanoPAN 5375

La figura 3.13 muestra el módulo RF de este *kit*, que es un transmisor que trabaja a 2.4 GHz. Este transmisor puede trabajar en tres modos de transmisión distintos, *HC Ranging*, *LD Ranging* y *R Comm*, según el ancho de banda y la velocidad con que se proporcionan los datos. Sus características son mostradas en la tabla 3.11.

Figura 3.13: Módulo transmisor RF nanoPAN 5375

Cuadro 3.11: Características de la placa del *kit* de desarrollo nanoPAN 5375

Tasa de datos	250 kbps, 1 Mbps
Rango de temperaturas ambientes	-40 a 70 ^{o}C
Voltaje de alimentación	2.5$\pm$0.2 V
Corriente de transmisión	típicamente 210 mA
Modo *HC Ranging* (80 MHz, 1 Mbps)	
Sensibilidad de RX	típicamente -89 dBm
Corriente RX	típicamente 51 mA
Potencia de salida de TX	típicamente +20 dBm
Rango de potencia de TX	típicamente 35 dB
Modo *LD Ranging* (80 MHz, 250 kbps)	
Sensibilidad de RX	típicamente -95 dBm
Corriente RX	típicamente 69 mA
Potencia de salida de TX	típicamente +20 dBm
Rango de potencia de TX	típicamente 35 dB
Modo *R Comm* (22 MHz, 250 kbps)	
Sensibilidad de RX	típicamente -96 dBm
Corriente RX	típicamente 34 mA
Potencia de salida de TX	típicamente +20 dBm
Rango de potencia de TX	típicamente 35 dB
Impedancia de carga de la antena	50 ohms nominales
Frecuencia de reloj SPI	hasta 24 Mbps
Frecuencia de Reloj de Tiempo Real	32.768 KHz
Cifrado hardware	128 bits
Certificaciones	FCC (ETSI en progreso)

3.6. Intel NUC5i7RYH

Intel NUC es un mini ordenador con la potencia de un equipo de sobremesa, con características para el entretenimiento, los videojuegos y la productividad en un formato de 10x10 cm. Cuenta con la quinta generación del procesador Intel Core i7-5557U. Tiene una tapa intercambiable que permite ampliar las prestaciones del sistema. Además está dotado de un disco duro de 2.5 pulgadas o un SSD M.2 además de Wi-Fi 802.11 ac, Bluetooth y sonido envolvente 7.1. El modelo en concreto utilizado es el NUC5i7RYH, que es el de la figura 3.14.

Figura 3.14: Ordenador de abordo Intel NUC5i7RYH

Especificaciones y características técnicas

Las características de la tabla 3.12 son las del ordenador de abordo utilizado.

Cuadro 3.12: Características técnicas Intel NUC 5i7RYH y otras características

Procesador	5ª generación del procesador Intel COre i7-5557U (3.1 GHz hasta 3.4 GHz con tecnología Intel Turbo Boost, doble núcleo, 4MB de caché, 28 W de potencia de diseño térmico (TDP))
Memoria	SODIMM DDR3L de canal doble 1.35 V, 1333/1600 MHz, 16 GB como máximo
Gráficos	Gráficos Iris 6100 1 Mini HDMI 1.4a 1 Mini DisplayPort 1.2
Sonido	Sonido envolvente hasta 7.1 vía Mini HDMI y Mini DisplayPort Conector para auriculares y micrófono en el panel frontal
Conectividad de Periféricos	2 puertos USB 3.0 en el panel posterior 2 puertos USB 3.0 en el panel frontal (1 para carga) 2 puertos USB 2.0 internos vía colector Sensor de infrarrojos en el panel frontal
Almacenamiento	Soporte interno para tarjeta M.2 SSD (22x42, 22x60 o 22x80) Soporte interno SATA3 para HDD/SSD de 2.5 pulgadas (hasta 9.5 mm de grosor)
Redes	Conexión de red Intel PRO/10/100/1000 Intel Wireless-AC 7265 M.2 soldada, antenas inalámbricas (IEEE 802.11ac, Bluetooth 4, Intel Wireless Display)
Caja	Plateada con tapa negra y corte por diamante en la parte superior De aluminio y plástico, dimensiones: 115 mm x 111 mm x 48.7 mm
Adaptador de corriente	Adaptador de AD-DC de 19 V, 65 W con montaje para pared Enchufes internacionales (IEC tipos A/C/G/I)
Características adicionales	Soporte para otras tapas a sustituir por el usuario Colectores NFC y AUXPWR Certificados de SO: Windows 7 y 8.1 logo'd Compatibilidad con SO: Linux Soporte de montaje VESA y soporte para orificio de montaje Diseño de refrigeración activo para acústica baja Soporte para conector de seguridad Kensington Guía de integración Entrada de alimentación CC de 12 - 19 V Circuito de sensores de consumo para protección frente a cortes de suministro por sobretensión Garantía limitada por 3 años

3.7. Conclusión

A pesar de que el sensor velodyne es de tipo láser es evidente que no es un sensor láser convencional ya que permite obtener información de distancia en 3 dimensiones con un ángulo de visión horizontal de 360 grados y un ángulo de visión vertical de 40 grados con una precisión de apenas 2 cm en medidas de distancia de hasta 80 - 100 metros. El sensor visual, la cámara estéreo ZED, también proporciona información en 3 dimensiones que no sólo está formada por las imágenes rgb si no que además, al igual que el velodyne, proporciona datos de distancia en una nube de puntos. Adicionalmente a las distancias proporcionadas por estos dos dispositivos, también se dispone de este tipo de información por parte de los sensores de rango del *kit* de desarrollo nano-PAN, que devuelven sólo un dato de distancia y no una nube de puntos. Para complementar la estimación del movimiento del robot también se dispone de una unidad de medición inercial.

A modo de conclusión, toda la información descrita en este capítulo deja claro que este proyecto cuenta con sensores de alto nivel económico lo que se traduce en altas prestaciones. Evidentemente, con los *datasets* generados se desean cumplir exigencias estrictas en cuanto a los resultados obtenidos del mapeado y localización del UAV, aspirándose a errores de pocos centímetros.

Siguiendo con la estructura descrita en la sección 1.3, a continuación se va a proceder a explicar el segundo de los bloques principales de este proyecto, el diseño *software* del sistema.

Capítulo 4

Diseño software del sistema

4.1. Introducción

Como se ha comentado en el capítulo 2, una de las ventajas de ROS es que permite la interoperatividad entre software y hardware desarrollado por personas o entidades diferentes, facilitando su uso a ingenieros. Es necesario hacer una labor previa de investigación para encontrar una versión estable del *driver* de cada uno de los sensores compatible con ROS. Una vez encontrado, lo siguiente es su instalación y el análisis de cómo funciona para conocer las modificaciones que se deben realizar para conseguir el modo de funcionamiento deseado del sensor. Cuando se tienen todos los recursos que permiten poner a funcionar los dispositivos, lo siguiente es procesar los datos obtenidos por cada uno de ellos de la forma que se desee para conseguir el objetivo establecido, que en el caso de este proyecto es el mapeado y localización simultáneos.

En este capítulo se van a describir detalladamente los pasos seguidos en el desarrollo de las aplicaciones *software* empleadas para poner en funcionamiento a cada uno de los sensores descritos en el capítulo 3. Todas ellas se han desarrollado bajo ROS, tal y como se ha comentado en el capítulo 1. También se va a mostrar, en los casos en los que sea posible, los resultados de los primeros experimentos realizados con cada uno de los dispositivos con el objetivo de validar que efectivamente se reciben datos del sensor y que dichos datos son coherentes.

4.2. Velodyne LiDAR

Para poner a funcionar el láser 3D de este proyecto se ha utilizado el repositorio *velodyne* de ROS, que contiene los paquetes *velodyne_driver*, *velodyne_msgs* y *velodyne_pointcloud* [40]. Para instalarlos se ha ejecutado el siguiente comando:

```
$ sudo apt-get install ros-indigo-velodyne
```

El primero de ellos se encarga de ajustar ciertos parámetros para configurar la comunicación con el HDL-32E y poder recibir datos de él.

Los *topics* que publica este paquete son dos:

- *velodyne_packets*, con mensajes del tipo *velodyne_msgs/VelodyneScan* (definido en *velodyne_msgs*) que contienen los paquetes de datos recibidos del sensor

- *diagnostics*, con mensajes del tipo *diagnostic_msgs/DiagnosticStatus*, que contienen información de estado del dispositivo.

Los parámetros con los que se realiza la configuración son:

- *frame_id*, por defecto *velodyne*
- *model*, modelo del dispositivo ("32E", "64E", "64E_S2", "64E_S2.1","VLP16")

- *npackets*, número de paquetes a publicar por mensaje. Por defecto es un número sufuciente para una vuelta completa

- *pcap*,

- *rpm*, velocidad de rotación del dispositivo en revoluciones por minuto. Por defecto es de 600 rpm

- *port*, puerto UDP del que leer. Por defecto el 2368

- *device_ip*, dirección IP del dispositivo. Por defecto usa cualquier dispositivo

El paquete contiene solamente un nodo, *velodyne_node*, y un nodelet *driver_nodelet* que tiene los mismo parámetros, publica los mismos topics y realiza la misma función que el nodo.

Para comenzar a recibir datos del Velodyne HDL-32E es muy importante establecer primero la conexión mediante *ethernet* con el ordenador al que se vaya a conectar. Esto puede realizarse desde el terminal o bien directamente creando una nueva conexión desde las *configuraciones de Red* de Ubuntu. La dirección IP de esta nueva conexión deberá ser 192.168.3.255, y la máscara de red 255.255.255.0, tal y como viene descrito en el *Manual de Usuario*. Si la conexión se configura de forma correcta, se deberían empezar a recibir datos del sensor. Una forma de comprobarlo es ejecutando la siguiente línea de comandos:

```
$ rosrun velodyne_driver velodyne_node _model:=32E
```

De la misma forma es posible cambiar el valor de los parámetros mencionados anteriormente según se desee.

Una vez que se están recibiendo datos del sensor, el paquete *xsens_pointcloud* es el encargado de subscribirse al *topic* donde están siendo publicados esos datos para transformarlos a una nube de puntos.

Este paquete contiene dos nodos,*cloud_node* y *transform_node*, y dos *nodelets*,*CloudNodelet* y *TransformNodelet*, que realizan la misma función que su respectivo nodo.

Por una parte el nodo *cloud_node* se subscribe al *topic* donde el paquete *velodyne_node* publica los datos del sensor (*velodyne_msgs/VelodyneScan*), los transforma a mensajes del tipo *sensor_msgs/PointCloud2* y los publica en el *topic velodyne_points*.

Los parámetros a configurar por este nodo son:

- *max_range*, máximo rango a publicar. Por defecto es 130.0 metros
- *min_reange*, mínimo rango a publicar. Por defecto, 0.9 metros
- *calibration*, fichero *yaml* que contiene la información específica de calibración
- *view_direction*, ángulo central de visión en radianes. Por defecto es 0.0
- *view_width*, ángulo de anchura de visión en radianes en el plano XY. Por defecto 2π

A modo de ejemplo, la ejecución del nodo se ha ejecutado de la siguiente forma:

```
$ rosrun velodyne_pointcloud cloud_node _calibration:=my_calibration.yaml
```

El fichero *calibration.yaml* es el que contiene la información relativa a la calibración del dispositivo y debe estar contenido en la carpeta donde se encuentra el ejecutable. En caso contrario, se debe especificar la ruta donde se encuentra.

Este nodo puede ejecutarse sin especificar el fichero de calibración y entonces se obtendría una nube de puntos del sensor sin calibrar. Para generar este fichero previamente se ha de ejecutar el siguiente comando, que guarda los datos contenidos en el fichero *32db.xml* (proporcionado con la documentación del dispositivo) en *my_calibration.yaml*:

```
$ rosrun velodyne_pointcloud gen_calibration.py 32db.xml my_calibration.yaml
```

4.2.1. Primeras pruebas

El paquete *velodyne_pointcloud* viene con algunos ficheros con extensión *launch*, que permiten lanzar varios nodos a la vez. Se ha utilizado el fichero *32e_points.launch*:

```
$ roslaunch velodyne_pointcloud 32e_points.launch calibration:=/ruta/32db.yaml
```

La herramienta utilizada para visualizar la nube de puntos es *rviz*, ejecutando el siguiente comando:

```
$ rosrun rviz rviz
```

En ella se ha añadido una nube de puntos del tipo *PointCloud2* que se subscribe al *topic velodyne_points*. El *Fixed Frame* se corresponde con el *frame_id* de la cabecera del mensaje publicado en dicho *topic*. Los resultados obtenidos se pueden ver en la figura 4.1.

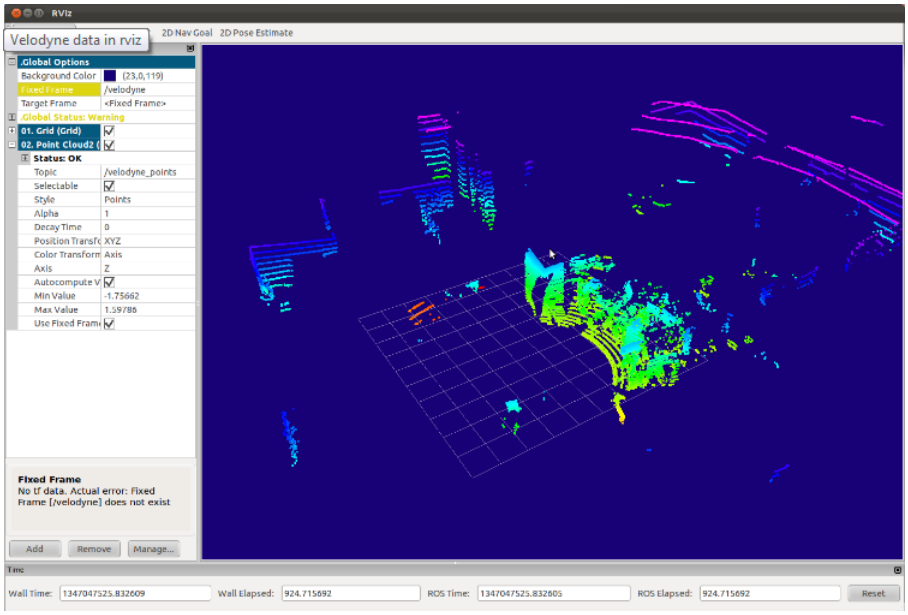


Figura 4.1: Visualización con rviz de la nube de puntos recibida del Velodyne LiDAR

Por otra parte, el nodo *transform_node* realiza la misma función que el nodo explicado anteriormente pero ofrece la posibilidad de especificar el marco de referencia (parámetro *frame_id*), que típicamente suele ser */map*. Aunque este nodo no ha sido utilizado, su ejecución se haría de la misma forma que el anterior:

```
$ rosrun velodyne_pointcloud transform_node _frame_id:=/map _calibration:=calibration.yaml
```

A continuación se va a proceder a explicar los paquetes de ROS utilizados para poner en marcha la cámara estéreo y se va a mostrar también las primeras pruebas realizadas con ella.

4.3. Cámara estéreo ZED

Para utilizar este sensor se han seguido los pasos aconsejados por la guía del usuario:

1. Instalación del software previamente requerido por el driver del sensor
2. Instalación del *driver*
3. Configuración de la cámara y carga de los archivos de calibración
4. Prueba de la cámara

4.3.1. Instalación del software previamente requerido por el *driver* del sensor

Para poder utilizar esta cámara con ROS es necesario tener instalado el software ZED SDK, que descarga los ficheros de calibración de la cámara conectada al ordenador. Este software necesita tener instalado una ver-

sión de CUDA igual o superior a la 6.5 y OpenCV versión 2.4.9 o superior.

Los comandos ejecutados para la instalación de CUDA han sido:

```
$ sudo apt-get remove nvidia-cuda-*
$ sudo dpkg -i cuda-repo-ubuntu1404_6.5-14_amd64.deb
$ sudo apt-get update
$ sudo apt-get install cuda
```

Y los siguientes para OpenCV:

```
$ sudo apt-get install build-essential
$ sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev
    libswscale-dev
$ sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev
    libjasper-dev libdc1394-22-dev
```

Estos tres comandos sirven para descargar los paquetes que son requeridos. Una vez descargado y descomprimido la versión apropiada de OpenCV, lo siguiente a realizar es ejecutar:

```
$ cd ~/opencv
$ mkdir release
$ cd release
$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D
    CMAKE_INSTALL_PREFIX=/directorio/donde/se/encuentra/el/archivo/de/OpenCV/descargado
$ make
$ sudo make install
```

Por último, se ha instalado la herramienta software ZED SDK proporcionada por el fabricante del dispositivo. Desde un terminal se ejecutaron los siguientes comandos:

```
$ cd /carpeta/contenedora/del/fichero/ZEDSDK-Linux-x86_64-vX.X.run
$ ./ZEDSDK-Linux-x86_64-vX.X.run
```

Tras ejecutar estos comandos se instalarán las herramientas *ZED Depth Viewer*, *ZED Explorer*, *ZED InstallChecker*, *ZED Settings App* y *ZED SVO Editor*.

4.3.2. Instalación del *driver*

La descarga del *driver* se ha realizado desde la página en Github [44], direccionada desde la página web del fabricante. El fichero descargado es un paquete de ROS llamado *zed-ros-wrapper-master*. Se ha copiado en la carpeta */scr* del directorio de trabajo de ROS y se ha compilado ejecutando el siguiente comando:

```
$ catkin_make
```

4.3.3. Configuración de la cámara y carga de los archivos de calibración

Uno de los medios para la configuración y calibración de la cámara es utilizando la herramienta software *ZED Settings App* que proporciona STEREOLABS [41], ya mostrada en la sección 3.2.1. Desde el terminal se puede ejecutar los siguientes comandos para lanzar la aplicación:

```
$ cd /carpeta/del/ejecutable/ZED Settings App
$ ./ZED\ Settings\ App
```

Por otra parte, alternativamente, una vez que se tiene instalado el driver del dispositivo en ROS se puede ejecutar el siguiente comando, que pondrá a funcionar la cámara. Es necesario tener acceso a Internet la primera vez porque al ejecutar el *driver* del dispositivo también lo hace el software ZED SDK, que descarga los archivos específicos de calibración de la cámara.

```
$ roslaunch zed_wrapper zed.launch
```

Al igual que sucedía con las aplicaciones software proporcionadas por el fabricante, desde este fichero se pueden configurar ciertos parámetros como por ejemplo *resolution*, *quality*, *sensing_mode*, *frame_rate*. Sus valores pueden ser importantes a la hora de producir retrasos de procesamiento de imágenes por la GPU del dispositivo al que esté conectada la cámara, como sucedió en las pruebas realizadas con la computadora *Jetson TK1* debido a la alta resolución de las imágenes.

Los *topics* que son publicados son:

- */camera/point_cloud/cloud*, nube de puntos (sensor_msgs/PointCloud2)
- */camera/depth/camera_info* información de la cámara (sensor_msgs/CameraInfo)
- */camera/depth/image_rect_color*, mapa de profundidades (sensor_msgs/Image)
- */camera/left/camera_info*, información de la cámara izquierda (sensor_msgs/Image)
- */camera/left/image_rect_color*, imagen rectificada a color de la cámara izquierda del dispositivo (sensor_msgs/Image)
- */camera/rgb/camera_info*, información de la cámara (sensor_msgs/CameraInfo)
- */camera/rgb/image_rect_color*, imagen rectificada a color (sensor_msgs/Image)

4.3.4. Primeras pruebas

El último paso es comprobar que la cámara funciona y que está calibrada. En primer lugar se ha hecho una prueba con el software ZED Depth Viewer, donde además de las imágenes rgb de ambas cámaras también se pueden ver el mapa de profundidades y la nube de puntos. La figura 4.2 muestra una prueba realizada con esta herramienta.

Se han ejecutado los siguientes comandos:

```
$ cd /carpeta/del/ejecutable/ZED_Depth_Viewer
$ ./ZED\ Depth\ Viewer
```

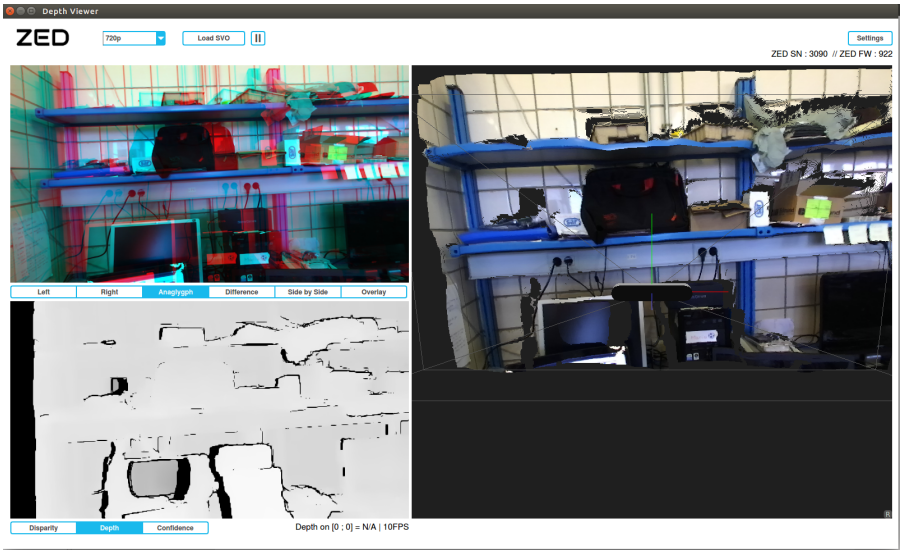


Figura 4.2: Primera prueba de la cámara estéreo. Uso de la aplicación software ZED Depth Viewer. La imagen izquierda superior se corresponde con las imágenes rgb de las cámaras izquierda y derecha, la imagen inferior se corresponde con el mapa de profundidades y la imagen derecha se corresponde con la nube de puntos 3D

Después de confirmar que los datos de la cámara recogidos con ZED Depth Viewer son correctos, se ha comprobado que los datos obtenidos con el paquete de ROS también lo son. Para ello se ha ejecutado el fichero *zed.launch* tal y como ha sido especificado anteriormente.

Para comprobar que las imágenes recibidas de la cámaras derecha e izquierda son correctas se ha lanzado el siguiente nodo dos veces, obteniéndose los resultados de la figura 4.3:

```
$ rosrun image_view image_view image:=/camera/left/image_rect_color
$ rosrun image_view image_view image:=/camera/right/image_rect_color
```

Figura 4.3: Imágenes recibidas de la cámara estéreo con la ejecución del *driver* bajo ROS. La imagen superior se corresponde con la cámara izquierda y la imagen inferior con la derecha

La verificación de la nube de puntos recibida se ha realizado ejecutando el siguiente nodo:

```
$ rosrun rviz rviz
```

Se ha añadido un elemento de tipo *PointCloud2* con el que *rviz* se subscribe al *topic /camera/point_cloud/cloud* y un elemento de tipo *Camera* con el que se subscribe a */camera/rgb/image_rect_color*. Es importante configurar correctamente el parámetro *Fixed_Frame* con el frame_id de la cámara, en este caso *zed_optical_frame*. Los resultados obtenidos son los mostrados en la figura 4.4.

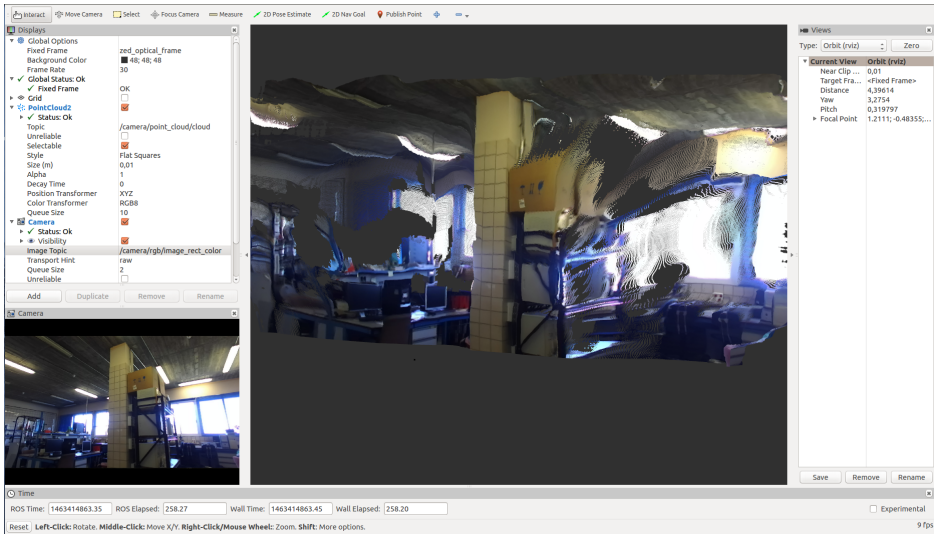


Figura 4.4: Visualización en *rviz* de la nube de puntos y de la imagen rgb

A continuación se va a proceder a explicar los paquetes de ROS utilizados para poner en marcha la unidad de medición inercial y se va a mostrar también las primeras pruebas realizadas con ella.

4.4. Unidad de medición inercial (IMU)

El paquete de ROS que se ha utilizado como driver para la puesta en marcha del MTi-G es *xsens_driver* [45]. Para instalarlo se ha ejecutado el siguiente comando:

```
$ sudo apt-get install ros-indigo-xsens-driver
```

Este paquete contiene tres nodos programados en lenguaje Phyton. El primer de ellos es *mtdef.py* que funciona a modo de librería para ser utilizada por el resto de nodos. El segundo de ellos es el *mtdevice.py*, que sirve para configurar la velocidad de transmisión, los datos deseados a la salida y el período de muestreo entre otros. El último de ellos es el *mtnode.py*, que se encarga de proporcionar los datos del sensor en mensajes de ROS.

El primer paso a realizar es ejecutar el nodo *mtdevice.py* para configurar el dispositivo. Si éste no se ejecuta, se tomará la configuración por defecto. Los comandos a ejecutar son:

```
$ cd /carpeta/del/ejecutable
$ ./mtdevice.py [Comandos] [Opciones]
```

En este caso, como los nodos están programados en lenguaje Python, el ejecutable será el mismo fichero donde se ha programado el nodo.

Los comandos pueden ser:

-h, –help : muestra la información que se va a detallar a continuación acerca de la configuración del dispositivo

-r, –reset : resetea el dispositivo a los valores por defecto de fábrica

-a, –change_baudrate=NEW_BAUD : cambia la velocidad de transmisión a NEW_BAUD

-c, –configure : configura el dispositivo (a continuación necesita los argumentos MODE y SETTINGS explicados más adelante)

-e, –echo : muestra los datos del MTi-G. Es el configurado por defecto si no es reemplazado por otro comando

-i, –inspect : muestra la configuración actual del dispositivo

-x, –xkf_scenario=ID : cambia el escenario del XKF

Las opciones pueden ser:

-d, –device=DEV : interfaz serie del dispositivo (por defecto /dev/ttyUSB0). Si su valor es 'auto' entonces se revisarán todos los puertos serie y el que tenga la velocidad adecuada será el utilizado

-b, –baudrate=BAUD : velocidad del puerto serie (por defecto 115200). Si es 0 entonces se probarán todas las velocidades hasta encontrar la apropiada

-m, –output_mode=MODE : selecciona la información de salida deseada.

- t : tempertura
- c : datos calibrados
- o : datos de orientación
- a : datos auxiliares
- p : datos de posición
- v : datos de velocidad
- s : datos de estado
- g : datos GPS sin procesar
- r : datos sin procesar

-s, –output_settings=SETTINGS : ajustes el dispositivo

t : contador de muestras

n : no contador de muestras

q : orientación en cuaternión

e : orientación en ángulos de Euler

m : orientación en matriz

A : aceleración calibrada

G : velocidad de giro calibrada

M : campo magnético calibrado

i : sólo entrada analógica 1

j : sólo entrada analógica 1

N : sistema de coordenadas de referencia North-East-Down en lugar de X North Z up

-p, –period=PERIOD : período de muestreo en (1/115200) segundos (por defecto 1152). El mínimo es 225 (1.95 ms, 512 Hz) y el máximo 1152 (10 ms, 100 Hz).

-f, –skip_factor=SKIPFACTOR : numero de muestras a omitir antes de enviar el mensaje *MTData*

A modo de ejemplo, si sólo se desearan datos de orientación en forma de matriz de rotación y de velocidad habría que ejecutar el siguiente comando desde la terminal:

```
$ ./mtdevice.py -c --output_mode=ov --output_settings=m
```

El siguiente paso sería lanzar el fichero *xsens_driver.launch* desde donde se ejecuta el nodo mtnode.py y se configuran los parámetros del MTi-G, como se muestra en la figura 4.5. Estos parámetros son el *frame_id*, el *frame_local* y el *frame_local_imu*.

```
<launch>
    <arg name="frame_id" default="/imu"/>

    <arg name="frame_local" default="NWU"/>
    <arg name="frame_local_imu" default="NED"/>

        <node pkg="xsens_driver" type="mtnode.py" name="xsens_driver" output="screen" >
                <param name="frame_id" value="$(arg frame_id)"/>

                <param name="frame_local" value="$(arg frame_local)"/>
                <param name="frame_local_imu" value="$(arg frame_local_imu)"/>
        </node>
</launch>
```

Figura 4.5: Contenido del fichero *xsens_driver.launch* del driver del MTi-G

Habría que utilizar el siguiente comando:

```
$ roslaunch xsens_driver xsens_driver.launch
```

Una vez ejecutado esto se debería tener el dispositivo en funcionamiento. Los *topics* que son publicados contienen mensajes ya definidos en ROS:

- /imu/data (sensor_msgs/Imu) : orientación, velocidad angular y aceleración lineal
- /fix (sensor_msgs/NavSatFix) : longitud, latitud y altitud (del GPS)
- /fix_extended (gps_common/GPSFix) : información del GPS más completa que la anterior
- /velocity (geometry_msgs/TwistStamped) : velocidad lineal y angular
- /magnetic (geometry_msgs/Vector3Stamped) : dirección del campo magnético
- /temperature (std_msgs/Float32) : temperatura

Para comprobar de forma gráfica si algunos de estos resultados son correctos se ha utilizado la herramienta gráfica de ROS *rviz*. En las imágenes 4.6 y 4.7 puede verse la orientación proporcionada por la unidad durante el experimento en interiores explicado en la sección 5.3.

```
$ rosrun rviz rviz
```

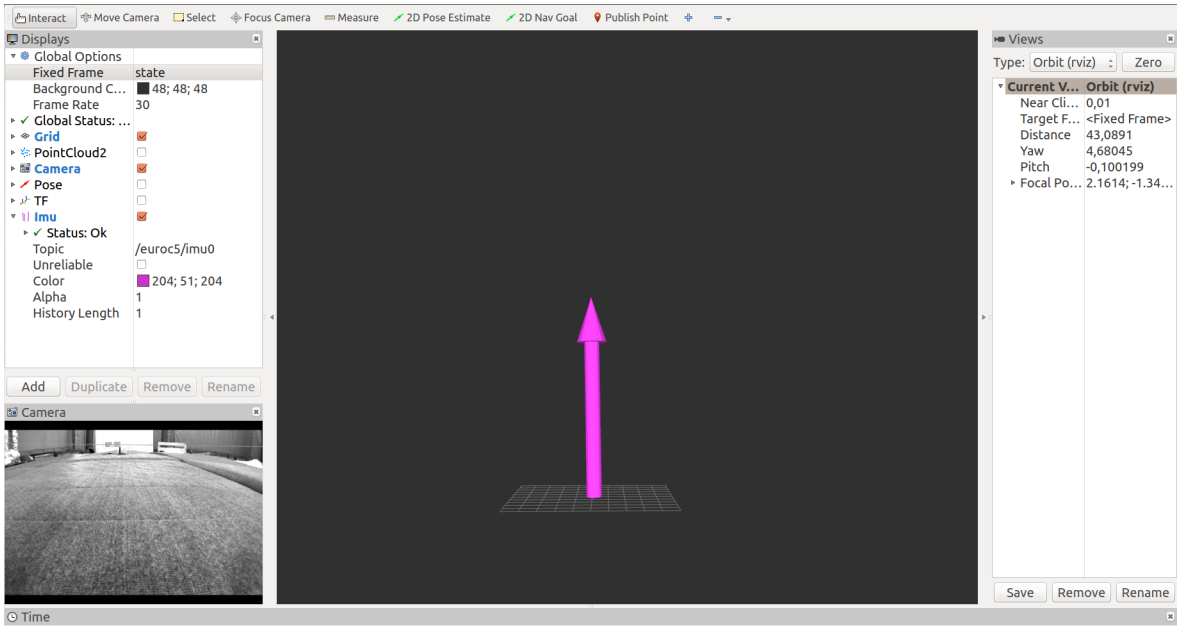


Figura 4.6: Representación de la orientación del UAV antes del despegue

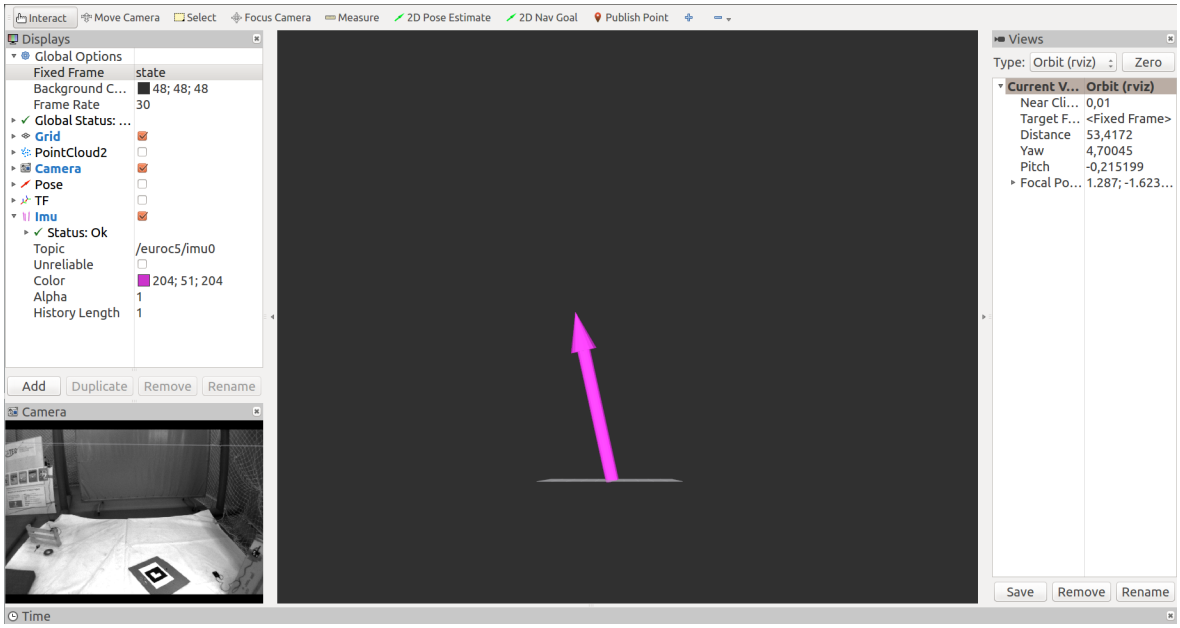


Figura 4.7: Representación de la orientación del UAV en un instante del vuelo

Adicionalmente, se ha creado un paquete para pasar la orientación de cuaterniones a ángulos. Para crear el paquete se han ejecutado los comandos:

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg euler_from_quaternion std_msgs rospy roscpp
```

El nodo principal de este paquete es *subscripterRPY_v2.cpp* cuya función es la de subscribirse al *topic* donde se publican los datos de orientación, transformarlos a ángulos y publicarlos en otro *topic* distinto.

La función principal de este nodo es la función *main*:

```
int main(int argc, char **argv) {
ros::init(argc, argv, "subscripterRPY_v2");
ros::NodeHandle nh;
Euler_pub = nh.advertise<euler_from_quaternion::Euler>("Euler_RPY", 1000);
sub = nh.subscribe("imu/data", 1000, cb);
ros::spin();
return 0;
}
```

Esta función lo que se hace es iniciar ROS indicándole el nombre del nodo y crea un publicador al *topic "Euler_RPY"* con un *buffer* de 1000 datos y un subscriptor al *topic "Euler_RPY"* que llamará a la función *"cb"* y que también tendrá un buffer de 1000 datos.

Los publicadores y subscriptores son declarados de la siguiente forma:

```
ros::Publisher Euler_pub;
ros::Subscriber sub;
```

La función *cb* es la siguiente:

```
void cb(sensor_msgs::Imu msg) {
tf::Quaternion q(msg.orientation.x, msg.orientation.y, msg.orientation.z,
    msg.orientation.w);
tf::Matrix3x3 m(q);
double roll, pitch, yaw;   // Roll, Pitch and Yaw in radians
double roll_deg, pitch_deg, yaw_deg; // Roll, Pitch and Yaw in Angles
double pi = 3.1415926535897; // 3.14159265358979323846
m.getRPY(roll, pitch, yaw);

euler_from_quaternion::Euler msgE;
// Conversion from radians to angles
roll_deg = (roll*360)/(2*pi);
pitch_deg = (pitch*360)/(2*pi);
yaw_deg = (yaw*360)/(2*pi);

msgE.roll = roll_deg;
msgE.pitch = pitch_deg;
msgE.yaw = yaw_deg;

Euler_pub.publish(msgE);
}
```

Esta función recibe como argumento un mensaje ya definido en ROS del tipo *sensor_msgs::Imu*. Posteriormente, utilizando la librería *tf/transform_datatypes.h*, crea un cuaternión *q* con las componentes *x*, *y*, *z* y *w* del elemento *orientation* de dicho mensaje. A partir de ese cuaternión crea una matriz *m* de orientación de la que consigue, finalmente, los ángulos de *Roll*, *Pitch* y *Yaw* (en radianes) con la función *getRPY*.

Esos datos de orientación ya en grados son introducidos en el mensaje *Euler* que es el que finalmente es publicado por el nodo. Este mensaje ha sido creado en la carpeta *msg* del paquete y contiene los siguientes componentes:

```
Header header

float64 roll
float64 pitch
float64 yaw
```

Para poder utilizar los dos tipos de mensajes detallados anteriormente es necesario incluir las librerías *sensor_msgs/Imu.h* y *euler_from_quaternion/Euler.h*.

El último paso a realizar es modificar los archivos *package.xml* y *CMakeLists.txt*. Como el nodo utiliza un mensaje que es creado expresamente para este paquete es necesario introducir la dependencia *message_generation* en el primero de ellos:

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

En el segundo es necesario añadir las siguientes líneas:

```
find_package(message_generation)

add_message_files(
FILES
Euler.msg)

generate_messages(
DEPENDENCIES
std_msgs)
```

Y, finalmente, para crear un ejecutable a partir del nodo *subscripterRPY_v2.cpp*, se añade:

```
add_executable(subscripterRPY_v2 src/subscripterRPY_v2.cpp)

add_dependencies(subscripterRPY_v2 euler_from_quaterinion_generate_messages_cpp)

target_link_libraries(subscripterRPY_v2
${catkin_LIBRARIES}
)
```

Cada vez que se vaya a ejecutar este nodo es necesario obtener datos del sensor por lo que se debe ejecutar también los nodos que se encargan de la comunicación con el MTi-G. Para poder lanzar la ejecución de estos dos nodos se ha creado el fichero *xsens_conversion.launch* en el paquete *euler_from_queaternion* que contiene los mismos datos que el fichero *xsens_driver.launch*, explicado anteriormente, y las siguientes líneas:

```
<node name="subscripterRPY_v2" pkg="euler_from_quaternion"
    type="subscripterRPY_v2.cpp" required="true"/>
```

Por último, se va a proceder a explicar los paquetes de ROS utilizados para poner en marcha los nodos de Nanotron.

4.5. Kit de desarrollo nanoPAN 5375 de Nanotron Technologies

El paquete de ROS que se ha creado para la puesta en marcha de los sensores de rango se ha llamado *interbeacon*. Este paquete contiene dos librerías, *cntronmod.h* y *cntronbase.h*. En la primera de ellas se define la clase *CNTronMod* y en la segunda todas las funciones de bajo nivel utilizadas para establecer la comunicación

con el nodo base y hacer que éste empiece a recibir datos del conjunto de dispositivos iguales que él que estén a su alcance.

Se ha desarrollado un nodo al que se le ha denominado *busca_nodo*. En él se ha creado un publicador que envía mensajes de tipo *interbeacon/measure* bajo el topic *"distances"*. Este mensaje no está ya definido en ROS, como sucede con los mensajes de tipo *geometry_msgs* por ejemplo, si no que ha sido creado específicamente en este paquete. Contiene dos campos:

```
int32[] id_destiny
float32[] measurement
```

El *id_destiny* es la identidad del sensor al que le corresponde el dato de distancia de *measurement*.

Lo siguiente es crear *nodoBase*, de la clase *CNTronMod*. A continuación se inicia la conexión con el puerto activado al conectar el nodo base al ordenador. Generalmente se suele denominar */dev/ttyUSB0*. En caso de que hubiera algún tipo de error, se mostraría en el terminal "Error de Inicializacion".

```
int main(int argc, char** argv) {

ros::init(argc, argv, "busca_nodo");
ros::NodeHandle n;
ros::Publisher chatter_pub = n.advertise<interbeacon::measure>("distances",
    1);

interbeacon::measure vector_dist;
float dist;
CNTronMod nodoBase= CNTronMod(1);

ros::Time ahora, despues;

int a = nodoBase.init("/dev/ttyUSB0");

if(a!=0){
cout<<"Error de Inicializacion\n";
return a;
}
```

Si el puerto se inicia correctamente, se procede a la comunicación con el nodo base que por defecto tendrá la identidad 255. Según el dato recibido se puede conocer si la lectura de la base es correcta o si, de lo contrario, se ha producido algún error. Se puede conocer el motivo de dicho error en los casos en que el dato recibido sea -1 y -5. Si el valor es 0, la lectura de la identidad de la base se habrá realizado satisfactoriamente.

```
int resultado=0;

/* ***** Test menu ***** */
resultado=nodoBase.readBaseID();
switch(resultado){
case -1:
cout<<"Error al enviar el comando. [ SendCommand() devuelve -6 ].\n";
break;
```

```
case -5:
cout<<"Error al encontrarse el buffer vacio. [ read() devuelve 0 ].\n";
break;
case 0:
cout<<"Lectura del ID de la base realizada con exito.\n";
cout<<"El ID de la base es "<< nodoBase.getBaseId()<<"\n";
break;
default:
cout<<"Error desconocido\n";
}
```

Si la lectura de la identidad de la base es correcta, se obtiene medida de distancia de todos los sensores desde el 1 hasta aquel definido en *NMAX_NODOS*. Los sensores que estén al alcance de la base irán rellenando con su identidad y la distancia los dos vectores del mensaje *vector_dist*, que es publicado cuando se completa el barrido de todos los dispositivos. El proceso seguirá realizando esto hasta que se reciba una señal de fin o error.

```
if(resultado == 0)
{
while(ros::ok())
{
vector_dist.id_destiny.clear();
vector_dist.measurement.clear();
//usleep(20);
//ahora = ros::Time::now();
for(int id=1; id<NMAX_NODOS; id++)
{
dist = nodoBase.measureTo(id);
if(dist > 0)
{
vector_dist.id_destiny.push_back(id);
vector_dist.measurement.push_back(dist);
}
}

chatter_pub.publish(vector_dist);
}
}
```

Cuando el proceso sale de este bucle, antes de finalizar, comunica por el terminal la identidad del nodo usado como base en caso de que éste no sea el fijado por defecto.

4.6. Conclusión

Continuando con la conclusión del capítulo anterior, las características de las medidas de los sensores vienen dadas por su diseño *hardware* y es evidente que las prestaciones que son capaces de proporcionar determinan la calidad de los resultados obtenidos. Sin embargo, de nada vale si se tienen sensores de gran calidad si los algoritmos y aplicaciones de la parte *software* del sistema están mal desarrolladas. Por ello, es imprescindible un buen desarrollo de las técnicas y métodos deseados no sólo para realizar la función para la que están

encomendados sino para mejorar aún más las prestaciones que proporcionan los sensores.

Siguiendo con la estructura descrita en la sección 1.3, a continuación se va a proceder a explicar el tercer y definitivo de los bloques principales de este proyecto, la realización de experimentos y la obtención de los *datasets*.

Capítulo 5

Experimentos

5.1. Introducción

Conocida la naturaleza de los sensores de que se dispone, su principio de funcionamiento y características técnicas y se consigue obtener datos coherentes de cada uno de ellos, se puede proceder a su integración en el robot y a la instalación del software necesario en el ordenador de abordo. Una vez que se consigue obtener datos con el UAV en vuelo lo siguiente es la realización de distintas pruebas que sirvan para grabar datos útiles a los que poder aplicar SLAM, y que permitan sacar conclusiones acerca de los movimientos que generan un mayor error en los resultados o los tipos de entornos que no son favorables para poder, así, buscar estrategias distintas que mejoren los resultados.

En este capítulo se van a describir los experimentos realizados para la obtención de *datasets* multi-sensor, el escenario elegido y sus características, el tipo de *hardware* y *software* utilizado y el contenido de los *bags* generados. Además se van a aplicar técnicas de SLAM, se van a mostrar resultados y se van a obtener conclusiones.

En primer lugar se va a exponer los experimentos en exteriores, realizados en las pistas de fútbol y baloncesto de la ETSI con una estructura que simula el movimiento del robot. El *hardware* utilizado ha sido la cámara ZED, la unidad de medición inercial y los sensores de rango descritos en el capítulo 3. Se van a mostrar los datos que contienen los *bags* generados. Finalmente se va a aplicar RGB-D SLAM a dichos datos y se van a mostrar resultados.

En segundo lugar se va a detallar las pruebas realizadas en interiores, que nacen de la participación de GRVC [46] en el concurso EuRoC [47] en colaboración con CATEC [48]. El objetivo es desarrollar tecnologías que posibiliten la colaboración de forma segura de robots aéreos con operarios en plantas de fabricación de aeronaves. El *hardware* utilizado está marcado por el concurso y no coincide con el detallado en este proyecto, aunque su naturaleza es la misma. Esta sección presentará la misma estructura que la anterior, en la que se describen las distintas pruebas realizadas, el escenario donde se han realizado y el contenido de los *bags* generados. También se van a mostrar resultados y conclusiones de la aplicación de un tipo de SLAM denominado SPTAM SLAM.

Finalmente se van a exponer una serie de conclusiones obtenidas a partir de las pruebas realizadas y de los

resultados del SLAM obtenidos.

5.2. Experimentos en exteriores

Los experimentos que se exponen a continuación tienen el objetivo de generar *bags* de datos que permitan una posterior navegación autónoma de un vehículo aéreo no tripulado mediante aplicación de técnicas de mapeado y localización simultáneas.

El hardware utilizado ha sido la unidad de medición inercial descrita en el proyecto, los nodos del *kit* de desarrollo nanoPAN 5375 y la cámara ZED. Todos estos sensores se han montado en una estructura con ruedas de aproximadamente un metro de alto, un metro de largo y menos de un metro de ancho que simula al robot. El ordenador de abordo utilizado ha sido el Intel NUC5i7RYH descrito en el capítulo 3.

El escenario elegido para grabar el *set* de datos se encuentra en las pistas de fútbol y baloncesto de la Escuela Técnica Superior de Ingeniería y se puede observar en las figuras 5.1, 5.2, 5.3, 5.4 y 5.5. Por las características y objetos del escenario se espera que el número de *features* sea abundante. En él hay un puente que tiene seis columnas, trípodes que han sido dispuestos para colocar nodos, varios bancos de metal, parte de dos alas de un avión real, unas escaleras y una portería de fútbol sala.

Figura 5.1: Imagen tomada desde el punto de partida, situado en la portería de fútbol. Es lo que el robot ve en la primera recta de la trayectoria

Figura 5.2: Visión del robot en el primer vértice de la trayectoria cuadrada

Figura 5.3: Parte del escenario que ve el robot en la segunda recta de la trayectoria

Figura 5.4: Visión del robot en la tercera recta de la trayectoria, desde el segundo vértice

Figura 5.5: Visión desde el tercer vértice. Es lo que ve el robot en la última recta de la trayectoria hacia el origen, situado en la portería de fútbol

El movimiento que sigue el robot en los experimentos es un cuadrado de aproximadamente 7.7 metros de lado. Se trazó la trayectoria de referencia en el suelo para intentar que se mantuviera en todas las pruebas. La velocidad a la que se mueve el robot es aproximadamente constante. La posición inicial es el vértice del rectángulo que está más próximo a la portería de fútbol, que también será la posición final para hacer el cierre del bucle o *loop closing*.

Se dispersaron por el escenario trece nodos que se dispusieron a distintas alturas pegados en paredes, escalones, bancos y trípodes. Sus identidades son el 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 17, 18, 23.

Se realizaron 4 pruebas, generando un *bag* por cada una de ellas. Se muestra su contenido utilizando la herramienta de ROS *rqt_bag*. Para lanzarla desde el terminal se debe ejecutar la siguiente línea de comandos:

```
$ rqt_bag
```

También se podría ejecutar el bag de la siguiente forma:

```
$ cd
$ cd /ruta/bag
$ rosbag play nombre.bag --clock
```

En la primera prueba el robot trazó una trayectoria de una sola vuelta al cuadrado. El tiempo total en realizarla fue de 127.434 segundos. El contenido del *bag* es el mostrado en la figura 5.6. La calidad de la imagen se redujo a VGA para evitar las latencias en el procesado además de para reducir el peso del *bag*. La figura 5.7 se corresponde con un determinado instante de tiempo del experimento en el que se reciben datos de todos los sensores de rango. Se puede ver sus identidades y sus valores.

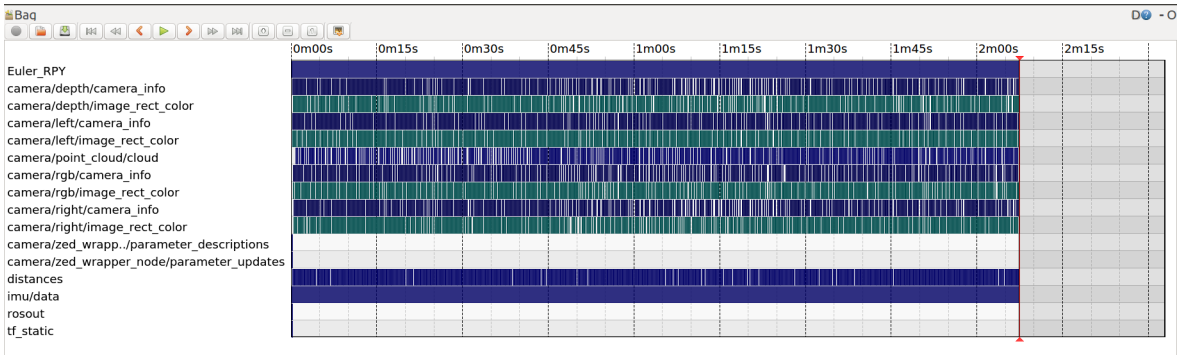


Figura 5.6: Contenido del *bag* generado en la primera prueba

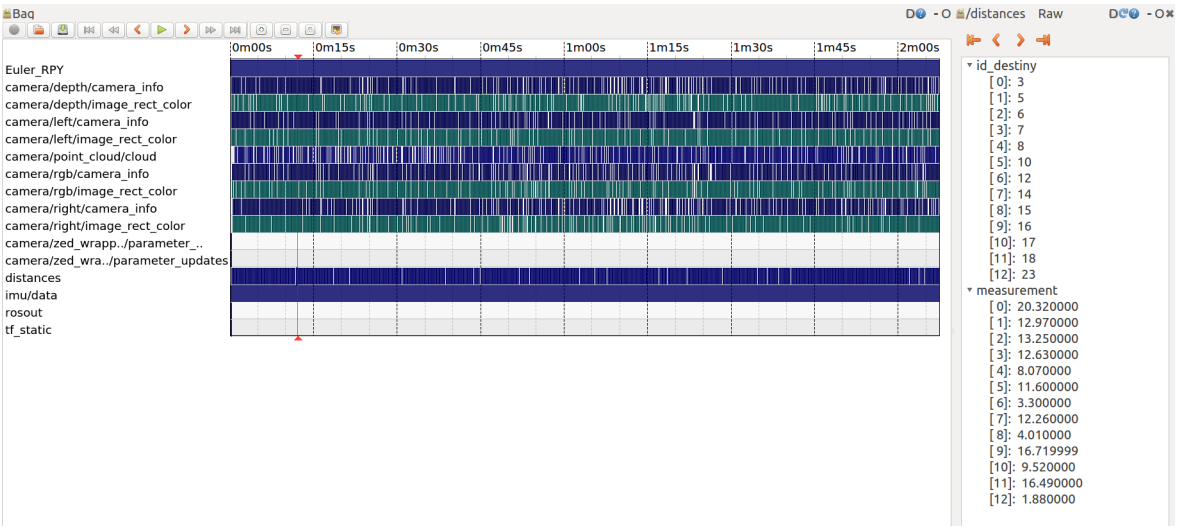


Figura 5.7: Identidades de los nodos de los que se reciben datos en un determinado instante de la trayectoria y la distancia a cada uno de ellos

El siguiente *bag* realizado tiene una duración de 184.90 segundos aproximadamente. Contiene los datos de una segunda prueba, en la que se realiza una trayectoria de dos vueltas con una velocidad relativamente mayor. Su contenido es el mostrado en la figura 5.8. La calidad de las imágenes sigue siendo VGA.

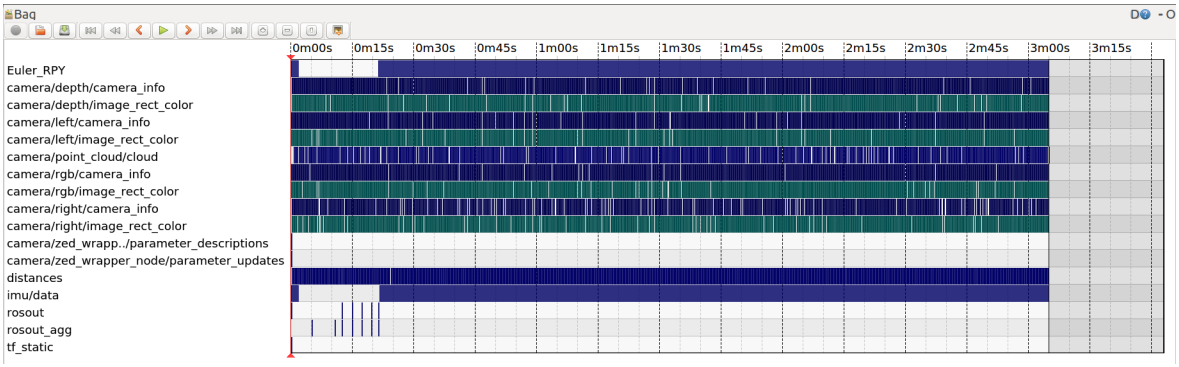


Figura 5.8: Contenido del *bag* generado en la segunda prueba

La tercera prueba duró 82.17 segundos aproximadamente. Se hizo lo mismo que en la primera pero el robot se mueve a mayor velocidad y la calidad de las imágenes es HD720. Su contenido puede verse en la figura 5.9. La evolución de los ángulos es la mostrada en la figura 5.10.

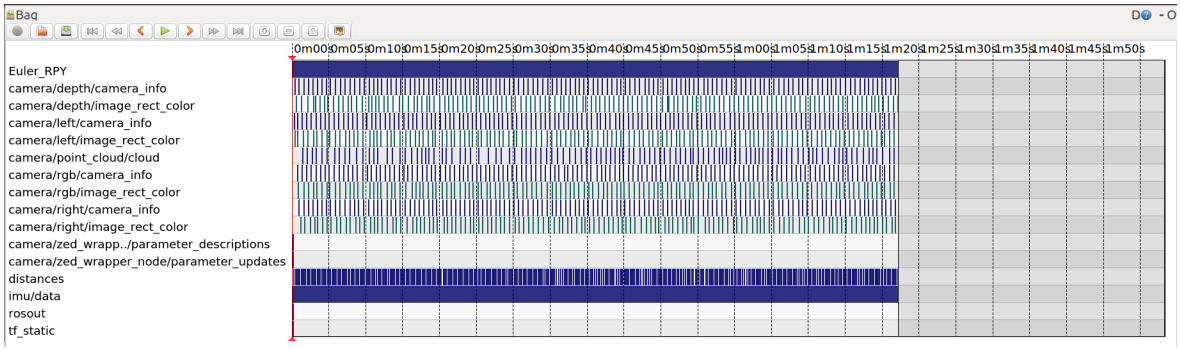


Figura 5.9: Contenido del *bag* generado en la tercera prueba

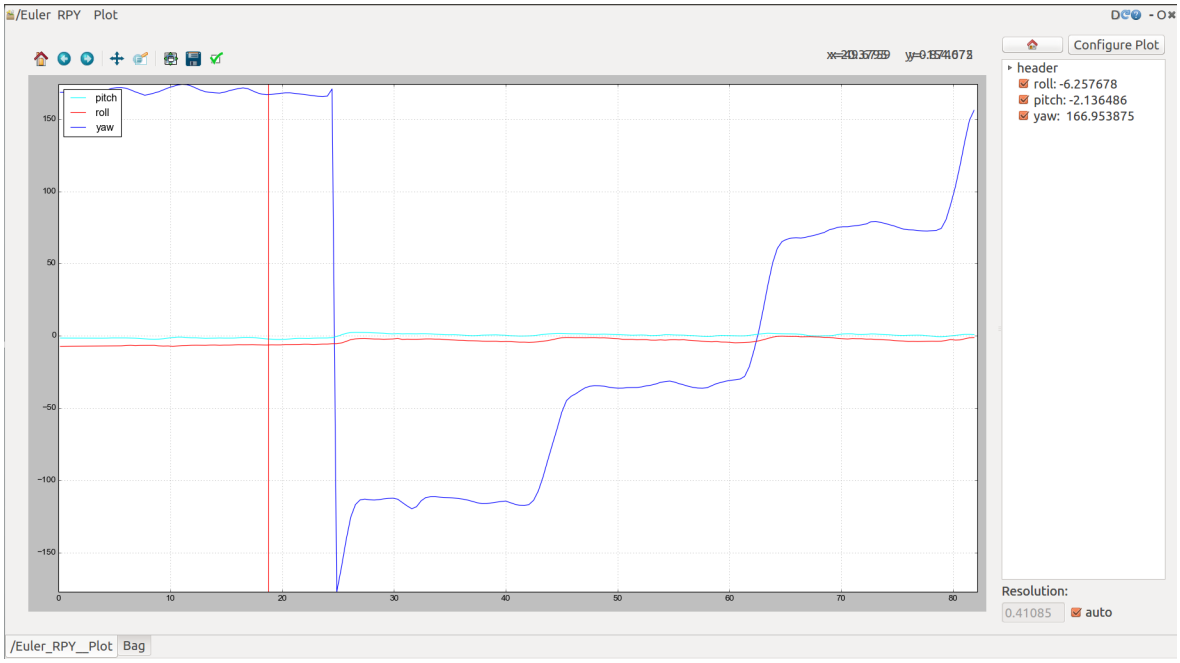


Figura 5.10: Representación de la evolución de los ángulos de *Roll*, *Pitch* y *Yaw* durante la tercera prueba

El cuarto y último *bag*, *cuarto.bag*, dura aproximadamente 135.17 segundos. Se trazó la misma trayectoria que la segunda prueba pero con mayor velocidad . Su contenido puede verse en la figura 5.11.

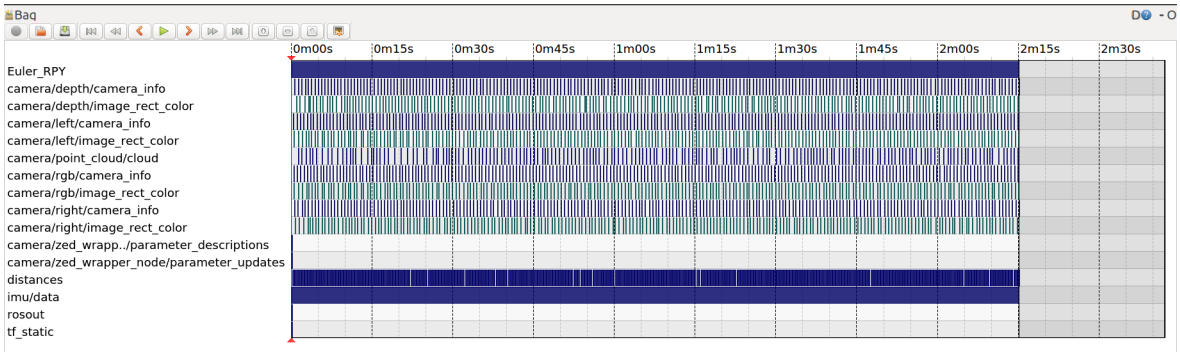


Figura 5.11: Contenido del *bag* generado en la cuarta prueba

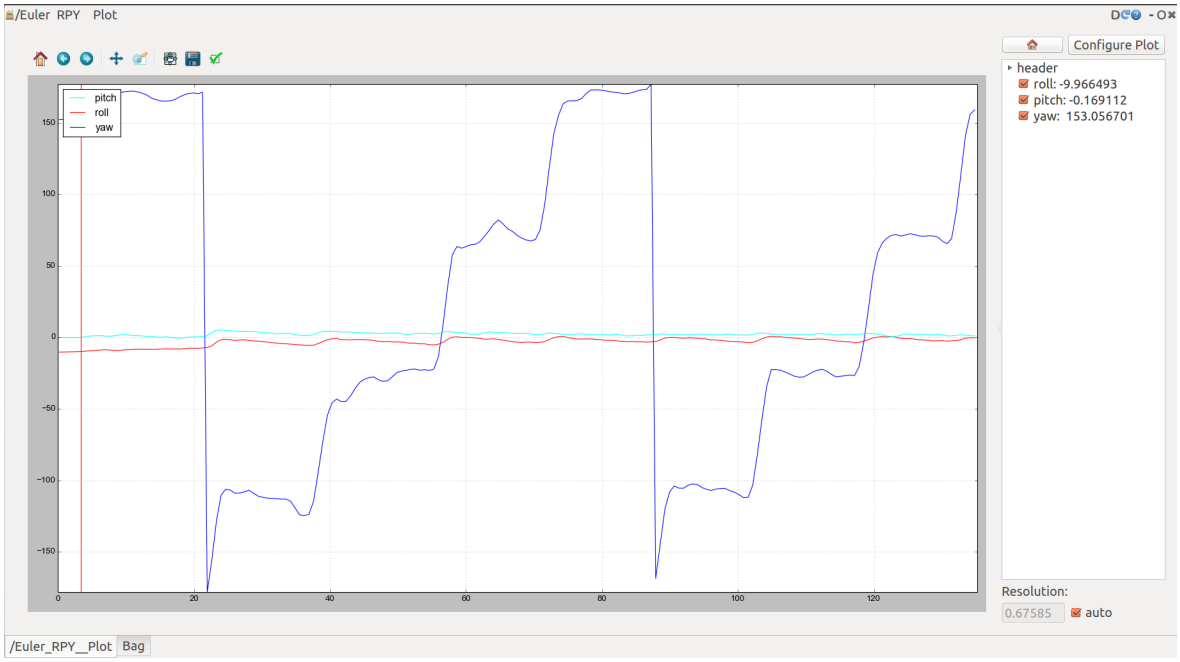


Figura 5.12: Representación de la evolución de los ángulos de *Roll*, *Pitch* y *Yaw* durante la cuarta prueba

Como se ha comentado, la calidad de las imágenes influye en el tiempo de procesado de la unidad pudiéndose llegar a producir cortes entre imágenes o retrasos. Estos pueden introducir errores en la estimación de la localización del SLAM, lo que lleva a que el *matching* de *features* entre *frames* de la cámara no se realice de forma correcta dando lugar a errores tanto en el mapa como en la localización. La ventaja de tener una mejor calidad de imagen es que los *markers* se detectan mejor.

El SLAM utilizado es un RGB-D SLAM ya que el sensor visual de que se dispone es de este tipo. El paquete de ROS utilizado ha sido [49], que devuelve tanto la posición del robot estimada como el mapa 3D en

forma de nube de puntos u OctoMapa [1]. Este paquete contiene un gran número de parámetros para ajustar el SLAM a las necesidades de cada usuario.

Se ha creado el fichero *modification_rgbdslam.launch*, que está basado en el fichero *rgbdslam.launch* del paquete original. En él se encuentran definidos todos los parámetros junto con una breve descripción de cada uno de ellos. Su contenido es el mostrado en la figura 5.13.

```
<launch>
  <node pkg="rgbdslam" type="rgbdslam" name="rgbdslam" cwd="node" required="true" output="screen">
    <!-- Input data settings-->
    <param name="config/topic_image_mono"              value="/camera/rgb/image_rect_color"/>
    <param name="config/topic_image_depth"             value="/camera/depth/image_rect_color"/>
    <param name="config/topic_points"                  value=""/> <!--if empty, poincloud will be reconstructed from image and depth -->
    <param name="config/camera_info_topic"             value="/camera/rgb/camera_info"/>
    <param name="config/wide_topic"                    value=""/> <!-- "Topics for stereo cam, e.g. /wide_stereo/left/image_mono" -->
    <param name="config/wide_cloud_topic"              value=""/>  <!-- "Topics for stereo cam e.g. /wide_stereo/points2" -->

    <!-- These are the default values of some important parameters -->
    <param name="config/feature_extractor_type"        value="ORB"/><!-- also available: SIFT, SIFTGPU, SURF, SURF128 (extended SURF), ORB. -->
    <param name="config/feature_detector_type"         value="ORB"/><!-- also available: SIFT, SURF, GFTT (good features to track), ORB. -->
    <param name="config/detector_grid_resolution"      value="3"/><!-- detect on a 3x3 grid (to spread ORB keypoints and parallelize SIFT and SURF) -->
    <param name="config/max_keypoints"                 value="600"/><!-- Extract no more than this many keypoints --> <!-- 600 -->
    <param name="config/max_matches"                   value="300"/><!-- Keep the best n matches (important for ORB to set lower than max_keypoints) --
> <!-- 300 -->

    <param name="config/min_sampled_candidates"        value="4"/><!-- Frame-to-frame comparisons to random frames (big loop closures) -->
    <param name="config/predecessor_candidates"        value="4"/><!-- Frame-to-frame comparisons to sequential frames-->
    <param name="config/neighbor_candidates"           value="4"/><!-- Frame-to-frame comparisons to graph neighbor frames-->
    <param name="config/ransac_iterations"             value="100"/>
    <param name="config/cloud_creation_skip_step"      value="2"/><!-- subsample the images' pixels (in both, width and height), when creating the
cloud (and therefore reduce memory consumption) -->


    <param name="config/cloud_display_type"            value="POINTS"/><!-- Show pointclouds as points (as opposed to TRIANGLE_STRIP) -->
    <param name="config/pose_relative_to"              value="largest_loop"/><!-- optimize only a subset of the graph: "largest_loop" = Everything
from the earliest matched frame to the current one. Use "first" to optimize the full graph, "inaffected" to optimize only the frames that were matched
(not those inbetween for loops) -->
    <param name="config/backend_solver"                value="pcg"/><!-- pcg is faster and good for continuous online optimization, cholmod and
csparse are better for offline optimization (without good initial guess)-->
    <param name="config/optimizer_skip_step"           value="1"/><!-- optimize only every n-th frame -->
  </node>
</launch>
```

Figura 5.13: Contenido del fichero usado para lanzar el RGB-D SLAM

Para poner a funcionar el SLAM hay que ejecutar la siguiente línea de comandos:

```
$ roslaunch rgbdslam modification_rgbdslam.launch
```

La ejecución del SLAM abre una ventana como la mostrada en la figura 5.14. En la imagen superior se muestran los resultados del mapa y de la trayectoria estimada del robot. En la imagen inferior izquierda se muestra la imagen a color a la que se subscribe el SLAM, en este caso */camera/rgb/image_rect_color*. La imagen que está a continuación es la nube de puntos obtenida de las imágenes estéreo, en este caso del *topic /camera/depth/image_rect_color*. En la siguiente imagen se muestra los *features* detectados de los que una parte formará el mapa de puntos y servirán como referencia para la localización del robot. Finalmente, en la imagen inferior de más a la derecha se representan flechas que simbolizan las distancia entre los distintos *features* que han sido detectados.

[1] Un OctopMapa es una estructura de mapeado 3D probabilista basado en Octetos

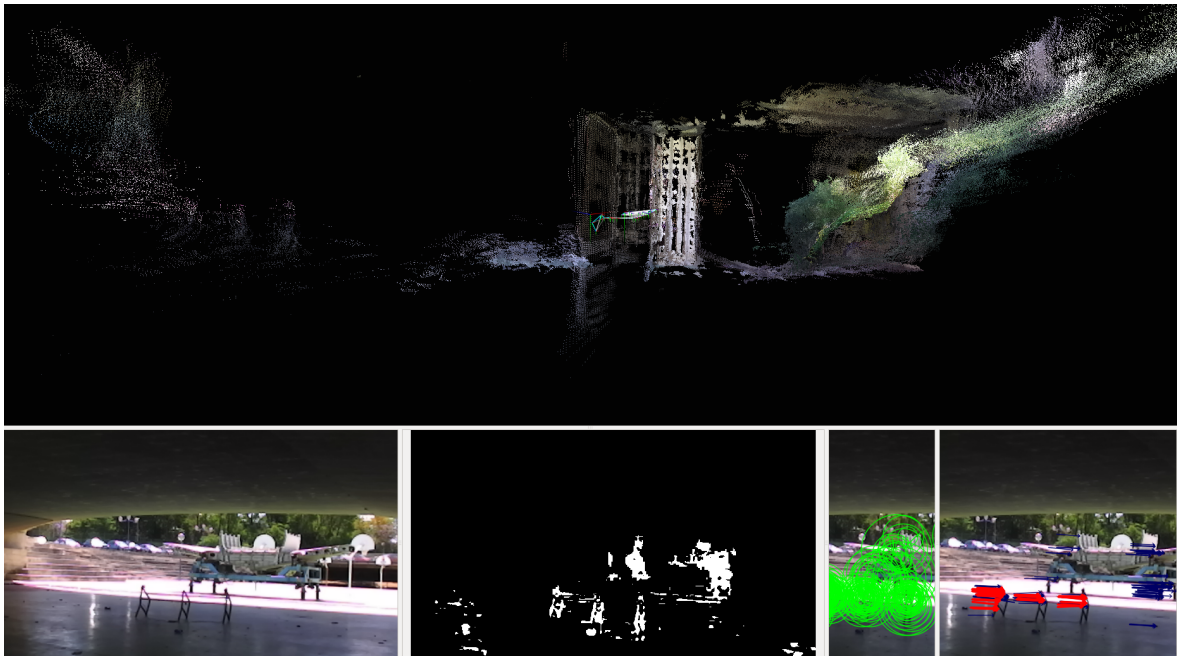


Figura 5.14: Imagen del resultado del RGB-D SLAM en ejecución

En las figuras 5.15, 5.17, 5.18, 5.19 se muestran partes de la nube de puntos del mapa generado con el primer *bag* de datos junto a una imagen rgb de la parte del escenario con la que se corresponde. Para demostrar que la nube de puntos está en 3 dimensiones en las imágenes 5.16a y 5.16b se muestra un ejemplo de una parte del mapa desde dos puntos de vista distintos.

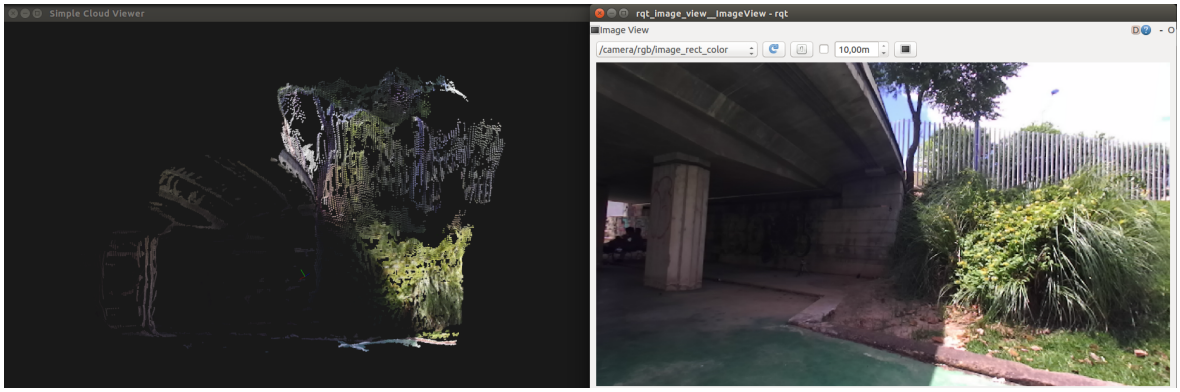
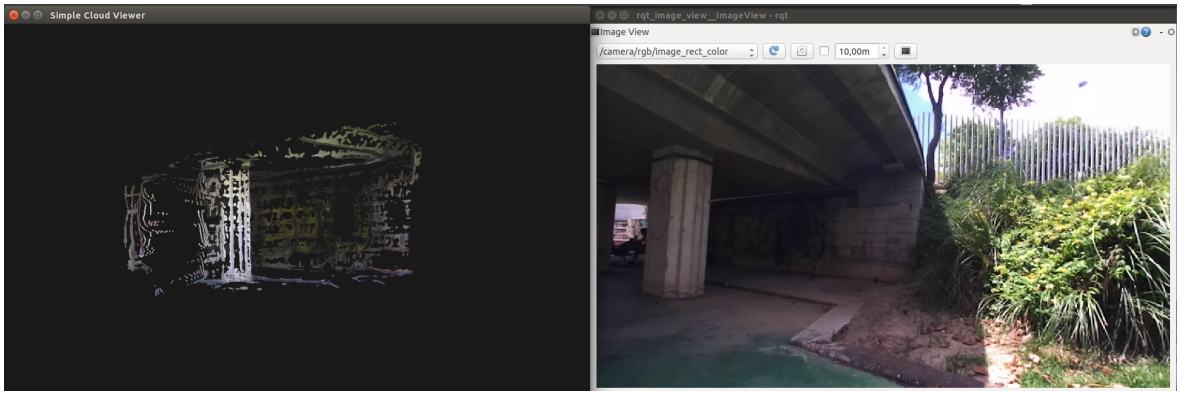
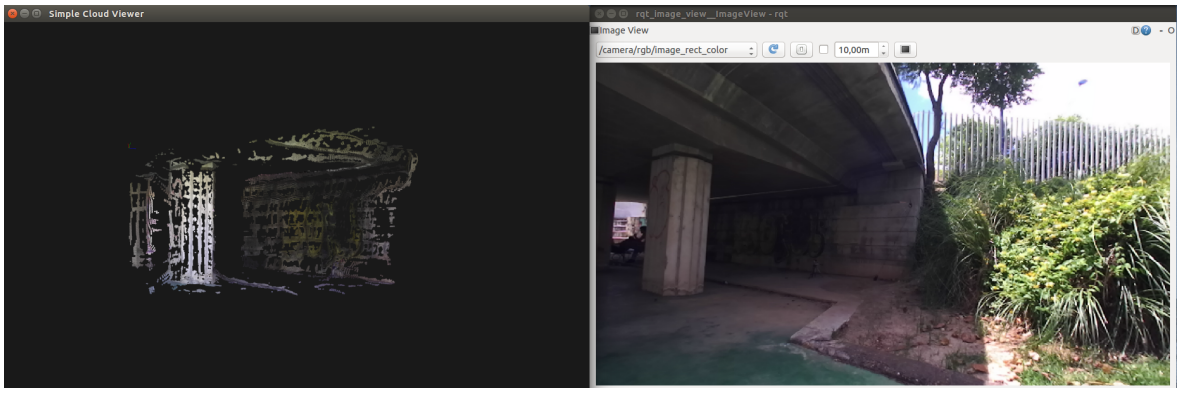


Figura 5.15: Parte de la nube de puntos del mapa generado e imagen ilustrativa del escenario real



(a)



(b)

Figura 5.16: Parte de la nube de puntos del mapa generado e imagen ilustrativa del escenario real

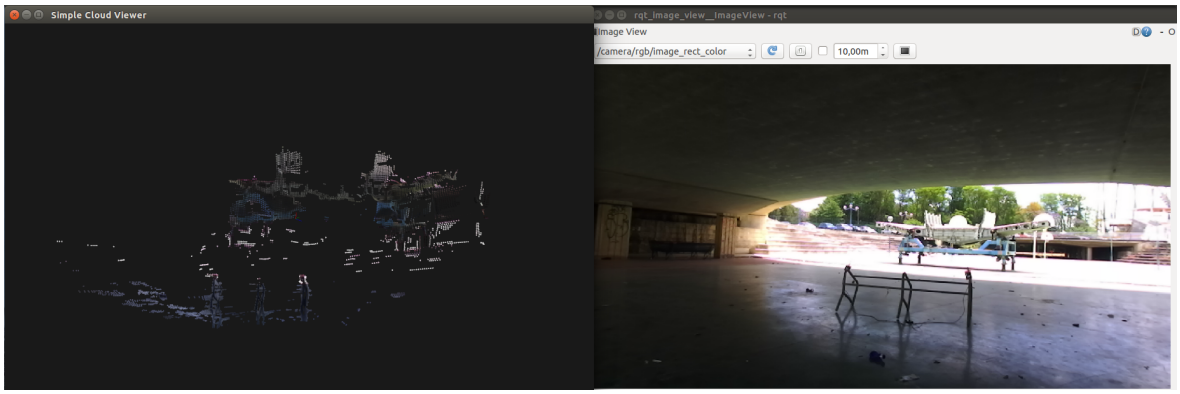


Figura 5.17: Parte de la nube de puntos del mapa generado desde dos puntos de vista diferentes e imagen ilustrativa del escenario real

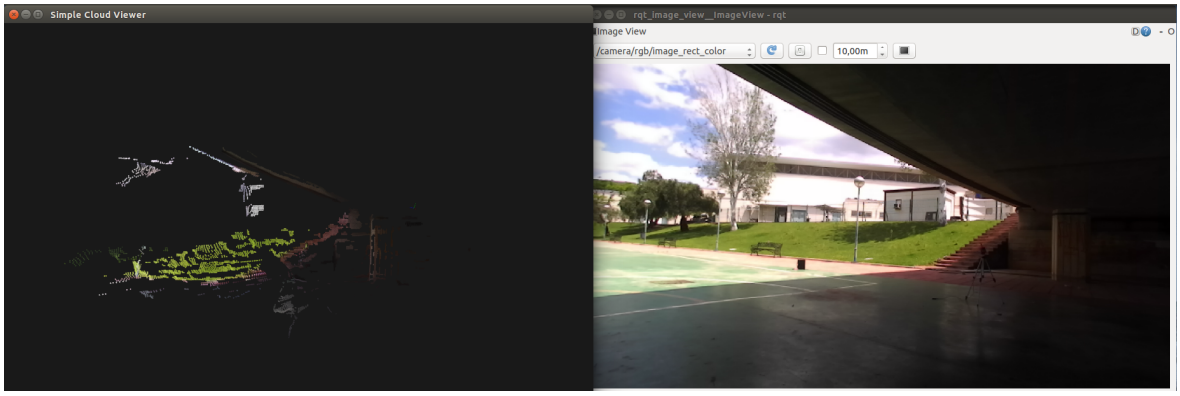


Figura 5.18: Parte de la nube de puntos del mapa generado e imagen ilustrativa del escenario real

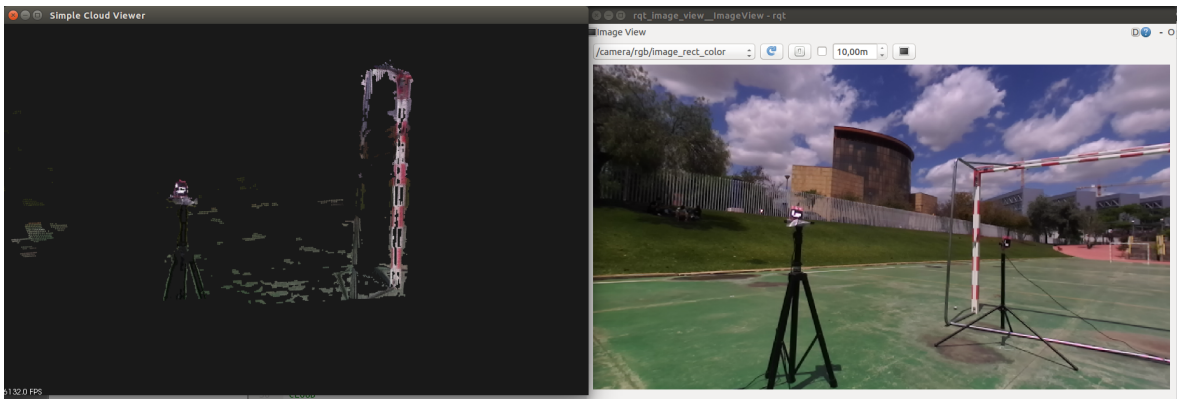


Figura 5.19: Parte de la nube de puntos del mapa generado e imagen ilustrativa del escenario real

Debido a los cambios de luz como los mostrados en la imagen 5.20 el SLAM en exteriores en estas pruebas realizadas no presentan datos satisfactorios, por ello no se muestra el mapa completo. Sin embargo en pruebas realizadas en interiores donde no había cambios bruscos de luz sí que se obtuvieron resultados satisfactorios.

Figura 5.20: Imagen resultante en un cambio brusco de luz

En la imagen 5.21 se muestra el resultado del mapeado y localización de la primera prueba, en un trayecto en el que no se producen cambios bruscos de luz como los comentados en el párrafo anterior. Por lo tanto, los resultados del SLAM son satisfactorios pero sólo en ciertos trayectos.

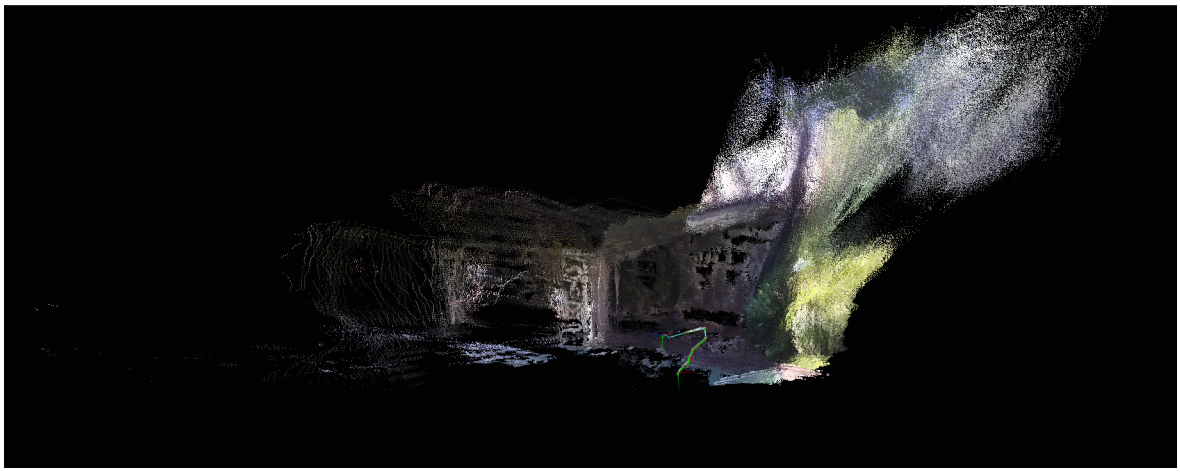


Figura 5.21: Resultado de SLAM de una parte de la trayectoria donde no se producen cambios bruscos de luz

5.3. Experimentos en interiores

Los experimentos que se exponen a continuación nacen como respuesta a la participación del Grupo de Robótica, Visión y Control [46] de la Universidad de Sevilla en el concurso EuRoC [47], *European Robotics Challenges*, en cooperación con el Centro Avanzado de Tecnologías Aerospaciales (CATEC) [48], un centro implantado y gestionado por la Fundación Andaluza de Desarrollo Aeroespacial (FADA) [50]. Los logos de

estas entidades son presentados en las figuras 5.22b y 5.22a, respectivamente.

(a) Grupo de Robótica, Visión y Control

(b) Centro Avanzado de Tecnologías Aerospaciales

Figura 5.22: Logos de las entidades que cooperan en la realización de los experimentos

EuRoc es una iniciativa propuesta por el sector de fabricación europeo para conseguir soluciones competitivas en robótica y así mantener el liderato global en este sector. En él se desean emprender tres retos o *challenges* industrialmente relevantes con aplicación futura, detallados en la figura 5.23.

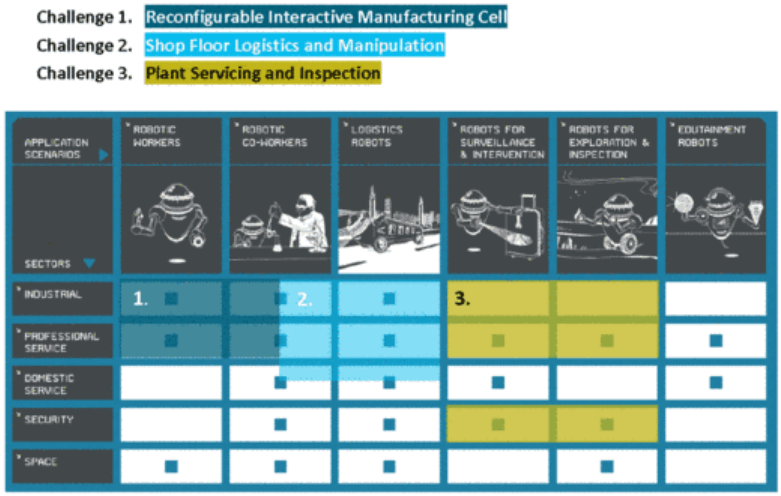


Figura 5.23: Retos a cumplir en el primero de los objetivos de EuRoC

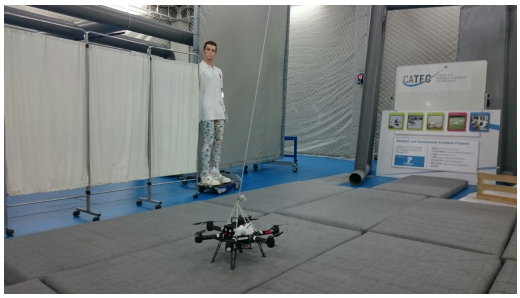
Los experimentos se han realizado en el *testbed* de CATEC. Es un espacio reducido, en interiores, donde existen objetos como tubos de PVC, cajas y carteles que hacen que se asemeje a una planta industrial como en la que se van a presentar los resultados finales. También se ha usado un maniquí para la simulación de un operario. Las figuras 5.24a, 5.24b, 5.25a y 5.25b son fotografías de este escenario donde se puede ver al UAV con el que se han realizado los vuelos. El motivo de poner colchonetas en el suelo es la obtención de una mejor odometría visual del movimiento ya que ésta se basa en la detección de puntos singulares del entorno y esto a veces era difícil de realizar por la homogeneidad del azul del suelo.

(a) Vista general del escenario de pruebas

(b) Fondo del escenario de pruebas

Figura 5.24: Fotografías del escenario de pruebas en el *testbed* de CATEC

(a) Lateral derecho del escenario de pruebas



(b) Lateral izquierdo del escenario de pruebas

Figura 5.25: Fotografías del escenario de pruebas en el *testbed* de CATEC

En cuanto al hardware, las pruebas se realizaron con nodos del *kit* de desarrollo nanoPAN 5375, una unidad de medición inercial y una cámara estéreo, todos ellos abordo del UAV mostrado en la figura 5.27. En este caso, la cámara utilizada es un sensor Inercial-Visual (VI-Sensor) [51] desarrollada por [52], que no es la descrita en el capítulo 3 aunque sigue siendo una cámara estéreo paralela que también proporciona las imágenes tanto de la cámara derecha como de la izquierda. Lo que la diferencia con la cámara ZED es que proporciona imágenes monocromas y no facilita directamente imágenes rectificadas ni el mapa de profundidades, aunque ambos pueden ser obtenidos con el paquete de ROS *stereo_image_proc* [53] en caso de que fuera necesario. En líneas generales esta cámara presenta características técnicas relativamente inferiores.

Figura 5.26: Cámara Visual-Inercial utilizada en estos experimentos

Figura 5.27: UAV proporcionado por EuRoC y con el que se han realizado los experimentos en interiores

Para la comparación de la estimación de la posición del robot realizada por el SLAM con su localización real se han utilizado varias cámaras infrarrojas Vantage de VICON [54] como la mostrada en la imagen 5.28. Estas pueden verse encendidas en las fotos del lugar de los experimentos, por ejemplo en la figura 5.24a.

Figura 5.28: Cámara infrarroja VICON utilizada para la captura del movimiento del robot

Se realizaron cinco vuelos diferentes y de distinta duración. Todos ellos contienen los mismos bags, que son los mostrados en la figura 5.29. En ellos falta la odometría visual, obtenida aparte mediante la ejecución de un paquete realizado por CATEC.

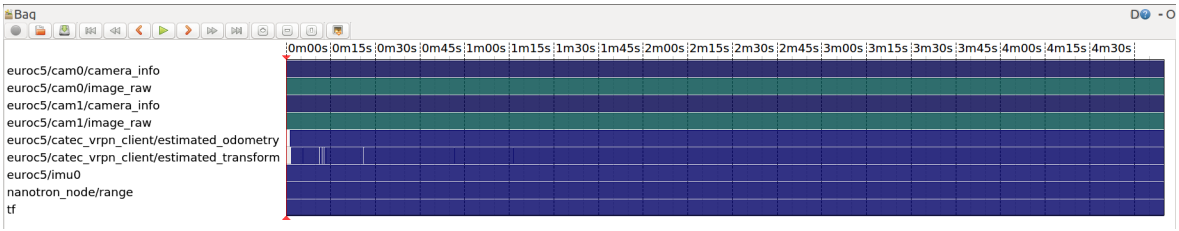


Figura 5.29: Contenido de uno de los *bags* generados de las pruebas en interiores

El paquete de SLAM que se ha utilizado es [55]. Está denominado SPTAM SLAM, del inglés *Stereo Parallel Tracking and Mapping*, un sistema capaz de realizar de forma paralela pero separada el problema de localización, de tiempo duro, y el problema de mapeado y refinamiento, que son menos restrictivas. El uso de una cámara estéreo permite, como en los experimentos realizados en exteriores, la construcción de un mapa 3D que tendrá una mayor precisión respecto de un SLAM monocular.

En el caso de no tener odometría visual, este nodo sólo necesita los *topics* de la información de las cámaras derecha e izquierda y de las imágenes rectificadas, obtenidas con el paquete de ROS *stereo_image_proc* comentado anteriormente. Sin embargo, en los experimentos aquí realizados sí que se dispone de odometría visual por lo que la estimación de la posición del robot no se realiza con el modelo de movimiento que contiene programado el paquete si no con la información que le proporciona esta odometría, obteniéndose mejores resultados.

Para la obtención de las imágenes rectificadas se ejecutó la siguiente línea de comandos. Lo que se hace es lanzar el paquete pero renombrando los *topics* a los que se subscribe:

```
$ ROS_NAMESPACE=/euroc5 rosrun stereo_image_proc stereo_image_proc
    /euroc5/left/image_raw:=/euroc5/cam1/image_raw /euroc5/right/image_raw:=/euroc5/cam0/image_raw
    /euroc5/left/camera_info:=/euroc5/cam1/camera_info
    /euroc5/right/camera_info:=/euroc5/cam0/camera_info
```

Para la ejecución del SLAM se ha realizado un fichero de extensión *launch* desde donde se lanza el ejecutable *sptam_node*, se renombran los *topics* que provienen de la cámara y se definen los parámetros de ejecución. El contenido de este fichero puede verse en la figura 5.30. Tal y como se puede observar, hay diversos parámetros que se pueden modificar para ajustar el funcionamiento del paquete al sistema sobre el que se quiera aplicar.

Los *topics* publicados son dos:

- point_cloud (sensor_msgs/PointCloud2): nube de puntos del mapa generado
- camera_pose (geometry_msgs/PoseStamped): posición de la cámara como resultado de la localización

```
<launch>

  <!-- Set use_sim_time true for datasets-->
  <param name="use_sim_time" value="true"/>
  <!-- Release -->
  <node pkg="sptam" type="sptam_node" name="sptam" output="screen" >

    <!-- Read S-PTAM parameters file -->
    <rosparam command="load" file="$(find sptam)/configurationFiles/visensor_caba.yaml" />

    <param name="approximate_sync" value="false" />
    <remap from="map_frame" to="world" />
    <param name="use_odometry" value="true" />
    <param name="odom_frame" value="state" /> <!-- viodom -->
    <param name="base_link_frame" value="imu0" /> <!-- state -->
    <param name="camera_frame" value="cam0"/>
    <param name="FeatureDetector/Name" value="GFTT"/>
    <param name="DescriptorExtractor/Name" value="BRIEF"/>
    <param name="DescriptorMatcher/Name" value="BruteForce-Hamming"/>
    <param name="DescriptorMatcher/crossCheck" value="false"/>
    <param name="MatchingCellSize" value="15"/>
    <param name="MatchingNeighborhood" value="1"/>
    <param name="MatchingDistance" value="25.0"/>
    <param name="EpipolarDistance" value="0.0"/>
    <param name="FrustumNearPlaneDist" value="0.1"/>
    <param name="FrustumFarPlaneDist" value="1000.0"/>

    <!-- Remap topics  -->
    <remap from="/stereo/left/image_rect"  to="/euroc5/left/image_rect"/>
    <remap from="/stereo/right/image_rect"  to="/euroc5/right/image_rect"/>
    <remap from="/stereo/left/camera_info"  to="/euroc5/cam1/camera_info"/>
    <remap from="/stereo/right/camera_info"  to="/euroc5/cam0/camera_info"/>

  </node>

</launch>
```

Figura 5.30: Imagen del contenido del fichero con extensión *launch* para la ejecución del SPTAM SLAM

Para lanzar el SLAM se debe ejecutar el siguiente comando:

```
$ roslaunch sptam euroc.launch
```

Una vez lanzado, se puede ver cómo se hace la detección de puntos característicos del escenario utilizados para localizarse con respecto a ellos en el mapa. Las imágenes 5.31, 5.32 y 5.33 muestran lo que en este párrafo se intenta explicar.

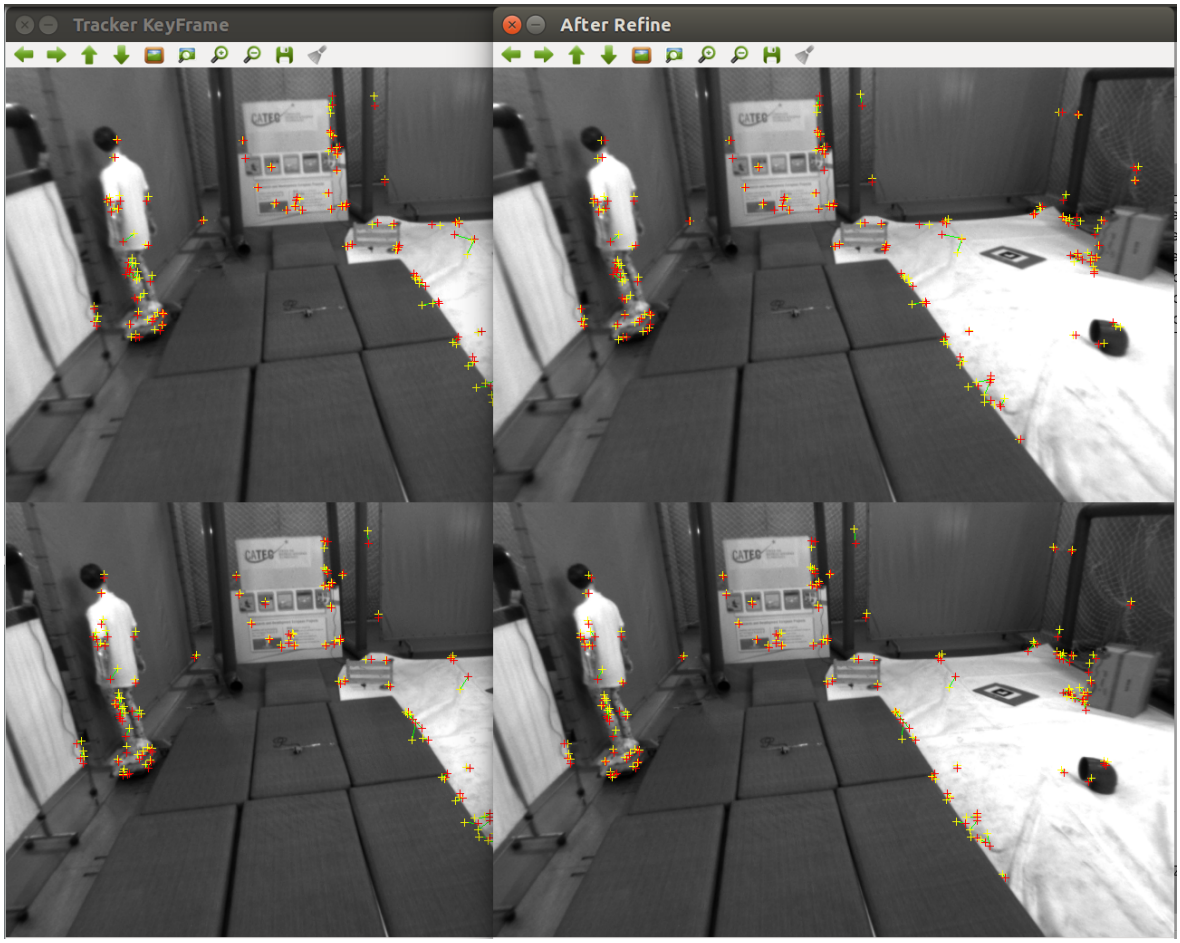


Figura 5.31: Visualización de la detección de puntos singulares o *features* del escenario en el *testbed* de CATEC

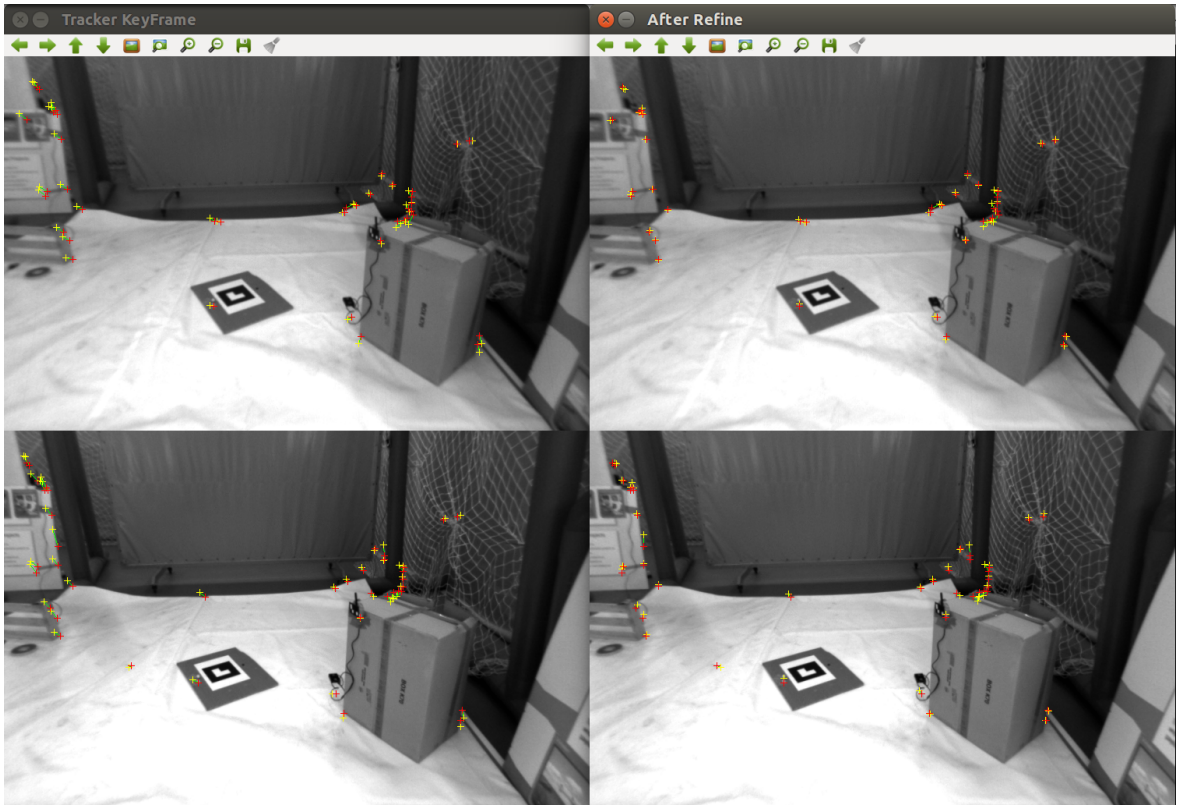


Figura 5.32: Visualización de la detección de puntos singulares o *features* del escenario en el *testbed* de CATEC

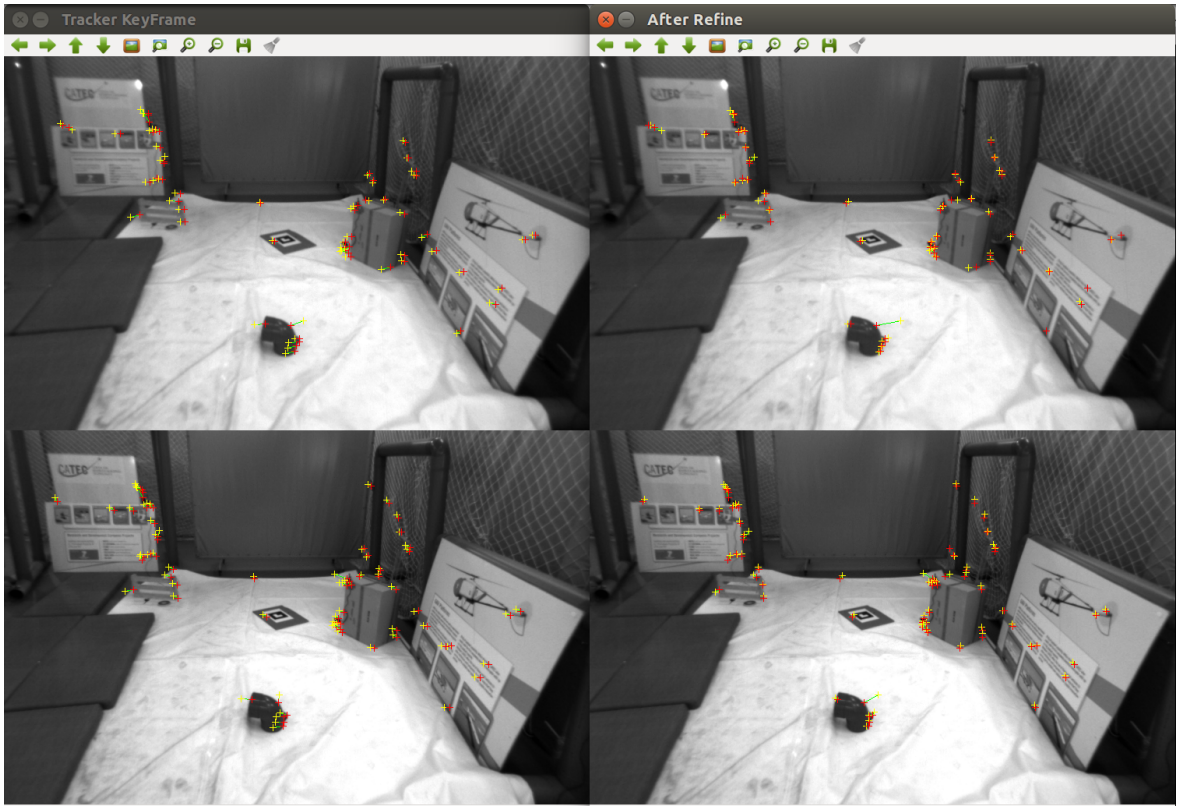


Figura 5.33: Visualización de la detección de puntos singulares o *features* del escenario en el *testbed* de CATEC

Utilizando la herramienta *rqt_plot* de ROS, se puede visualizar la posición del robot estimada según el SLAM (*/sptam/robot/pose_translated*), según la odometría visual (*/viodom_position*) y según el *ground truth* (*/euroc5/catec_vrpn_client*), que en este caso es la posición obtenida de las cámaras VICON.

Si se realiza el SLAM utilizando la odometría visual en lugar del modelo del movimiento del robot, se obtienen resultados mucho mejores. Se puede ver en las imágenes 5.34, 5.35 y 5.36. En el eje de abscisas se representa el tiempo en segundos y en el eje de ordenadas una coordenada determinada de la posición del robot en metros.

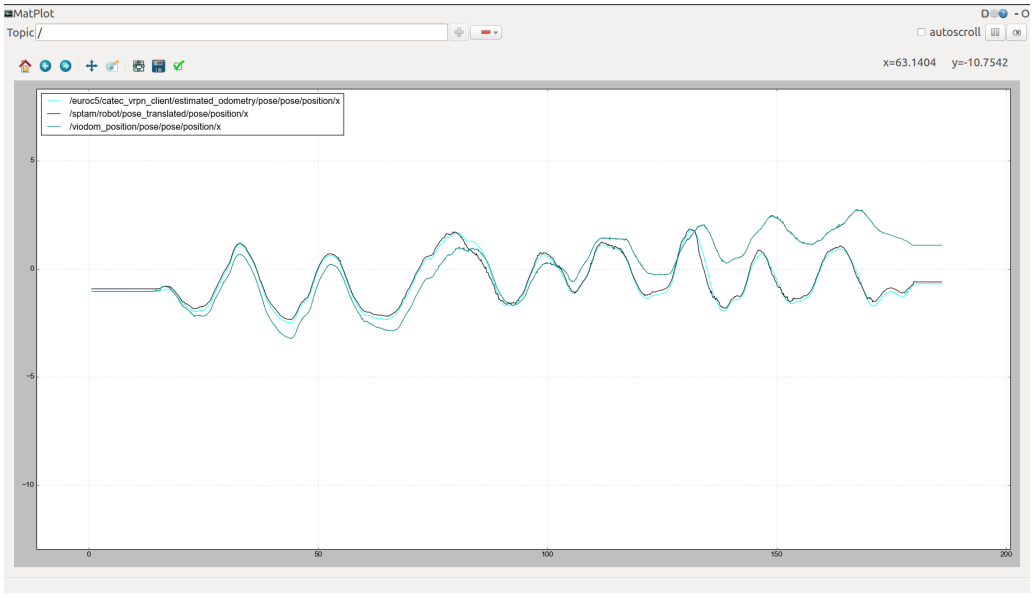


Figura 5.34: Representación de la localización en el eje x del robot del *ground truth*, del SLAM y de la odometría visual de la primera prueba

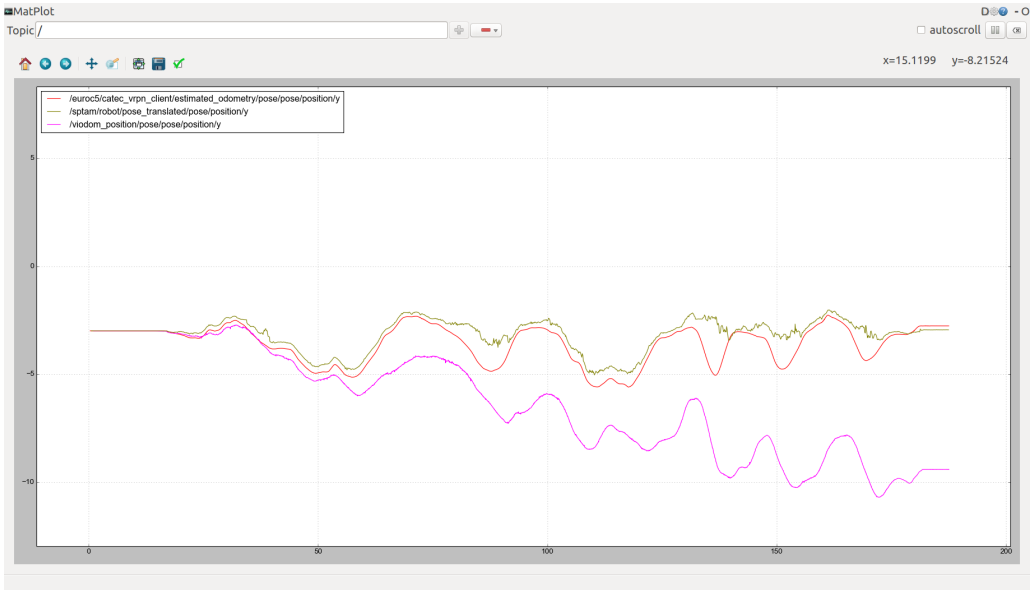


Figura 5.35: Representación de la localización en el eje y del robot del *ground truth*, del SLAM y de la odometría visual de la primera prueba

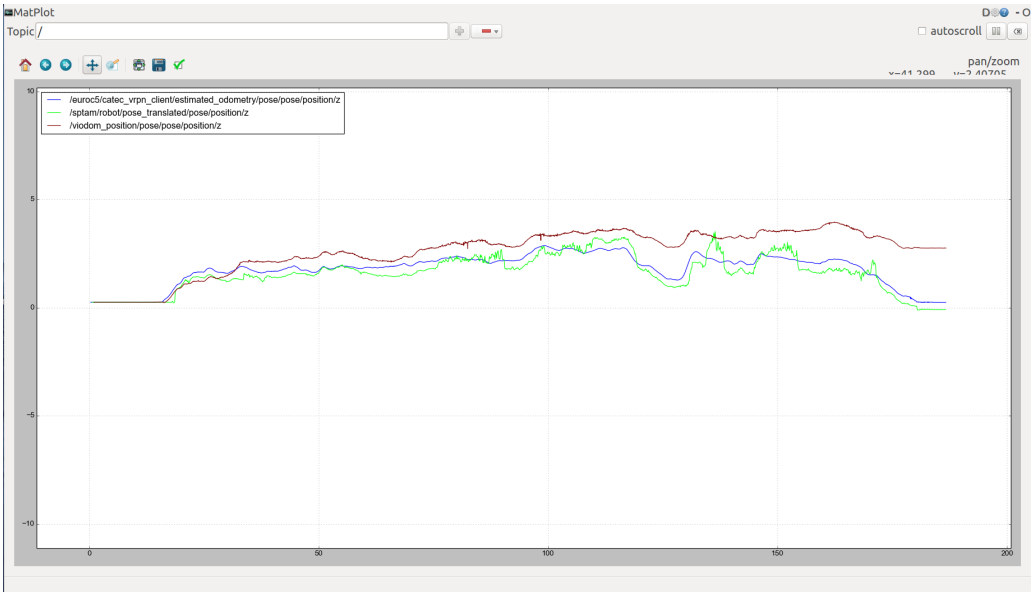


Figura 5.36: Representación de la localización en el eje z del robot del *ground truth*, del SLAM y de la odometría visual de la primera prueba

Observando las figuras anteriores se puede comprobar que la posición de la odometría visual va acumulando errores desde el inicio, por lo que al final del experimento presentará una deriva relativamente importante en la estimación. La posición del SPTAM SLAM se comporta mejor que la odometría visual, con un error medio en x de 22.6129 cm, en el eje y de 58.1185 cm, y en el eje z de 33.2175 cm, mejorando el problema de localización.

Las dos imágenes siguientes muestran los resultados obtenidos en otro vuelo. Se ha aplicado SLAM a los datos del mismo vuelo dos veces. Se compara el movimiento del robot estimado por el SLAM y la estimada por la odometría del VICON. En 5.37 el error medio en x es de 10.9691 cm, en y de 8.7483 cm y en z de 23.7872 cm. En 5.38 el error medio en x es de 8.1313 cm, en y es de 8.7529 cm y en z de 56.0087 cm.

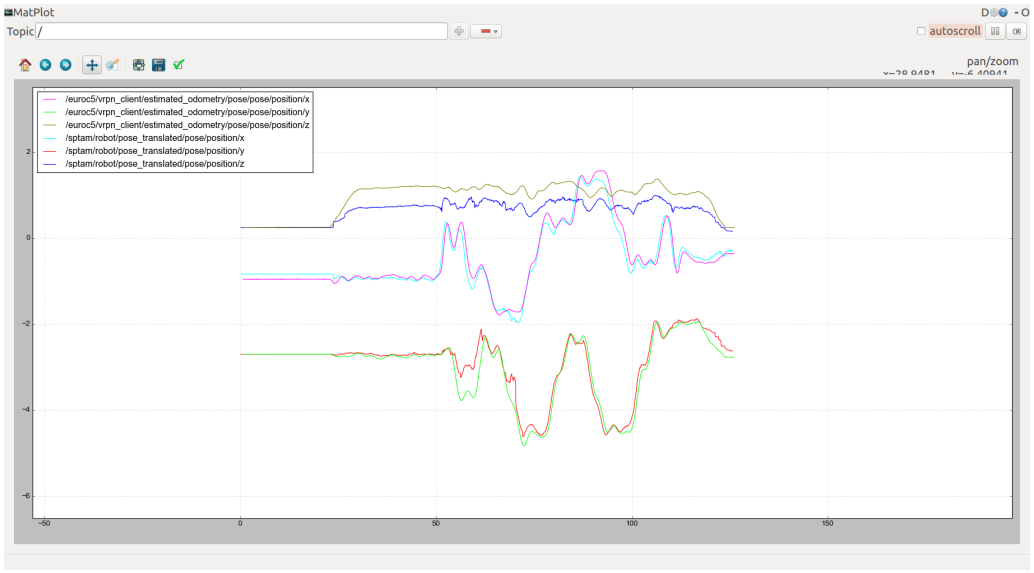


Figura 5.37: Representación de la estimación de la posición del robot realizada por el SLAM y por el *ground truth*

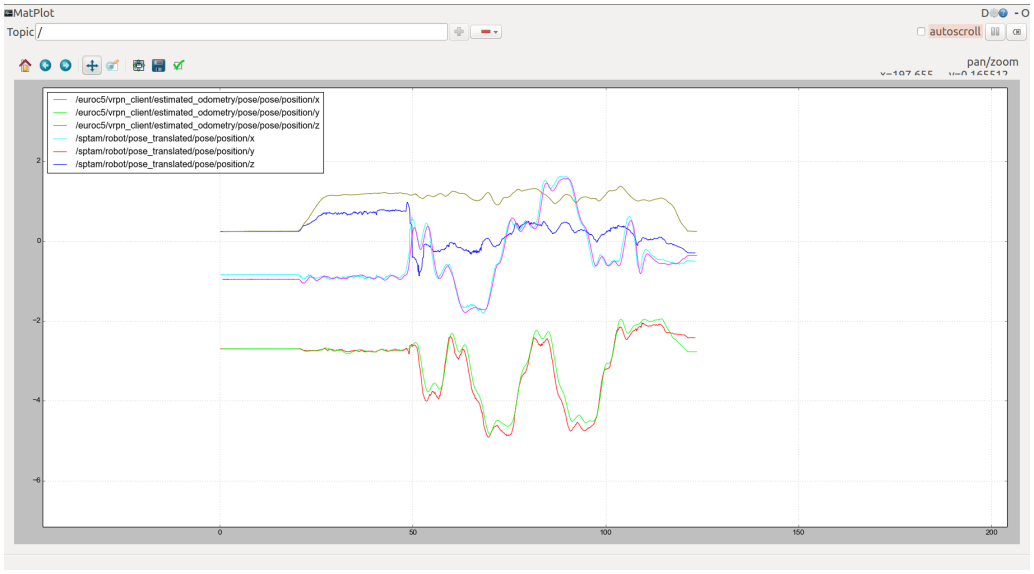


Figura 5.38: Representación de la estimación de la posición del robot realizada por el SLAM y por el *ground truth*

Se llega a la conclusión de que el algoritmo de SLAM tiene algo de aleatoriedad en sus estimaciones por lo que posiblemente esté basado en un filtro de partículas. El despegue influye negativamente en la posición del

robot, sobre todo en la coordenada z.

En la figura 5.38 se muestran los resultados obtenidos de otro vuelo diferente, en el que una vez despegado se realiza un vuelta sobre el eje z de 360 grados. Los resultados son negativos. Los errores obtenidos son mucho mayores que los de las pruebas previamente realizadas.

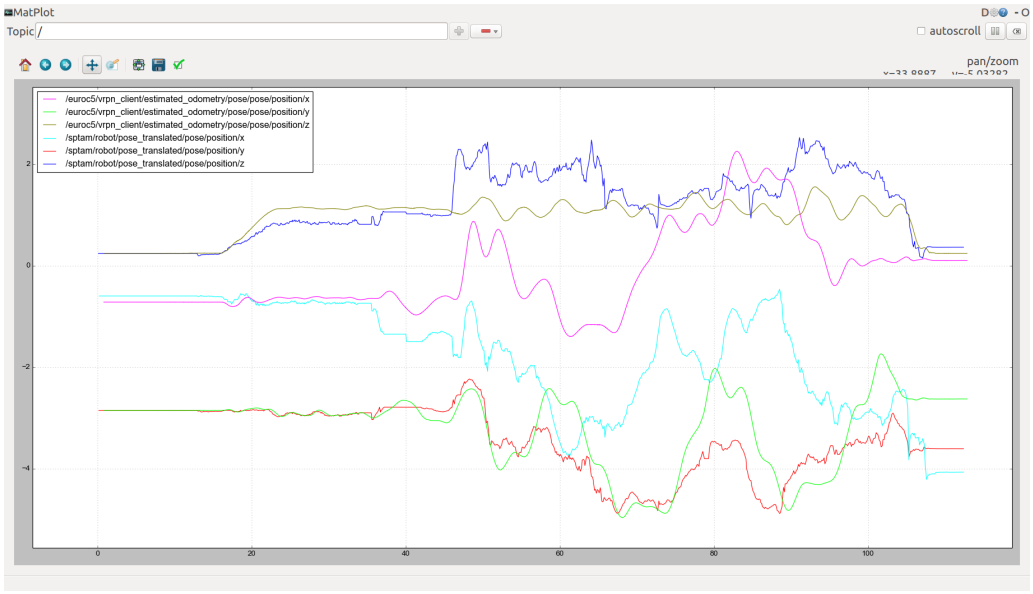


Figura 5.39: Representación de la estimación de la posición del robot realizada por el SLAM y por el *ground truth*

Volviendo al bag relativo a los resultados mostrados en las imágenes 5.37 y 5.38, se ha realizado un bag que contenga los mismos datos pero donde se haya quitado la parte del despegue para comprobar si la estimación realizada por el SLAM se mejora. Los resultados son mostrados en la imagen 5.40.

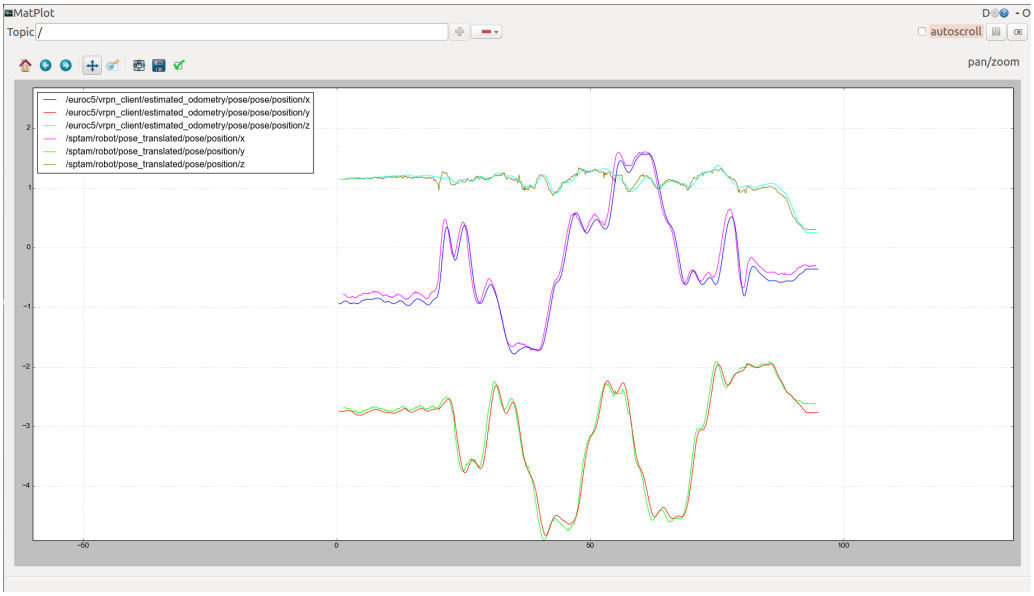


Figura 5.40: Representación de la estimación de la posición del robot realizada por el SLAM y por el *ground truth*

El error medio en x es de 8.13449 cm, el error medio en y es de 5.60957 cm, y el error medio en z es de 3.1664 cm. Por lo tanto, se llega a la conclusión de que el despegue es desfavorable en la estimación de la posición del robot. Como futura tarea queda pendiente la realización de pruebas donde el SLAM se lance una vez que el UAV haya despegado y buscar una estrategia de vuelo alternativa a realizar una vuelta de 360 grados.

5.4. Conclusión

Como se ha podido comprobar en este capítulo, una de las ventajas que ofrece ROS es el grabado de sets de datos que permiten ser reproducidos luego, facilitando así su uso en el desarrollo y prueba de aplicaciones en instantes posteriores.

Cuando los *datasets* deben contener datos de sensores visuales se ha de tener en cuenta ciertos factores, como son la calidad de las imágenes. Dependiendo de la resolución a la que esté configurada la cámara se deberá tener precaución en el tamaño del *bag* ya que si las imágenes ocupan un tamaño relativamente grande el registro del *bag* se llenará pronto y sólo se grabará una parte del experimento. Una alternativa es aumentar la duración y el tamaño del registro del *bag*, que puede detallarse mediante un argumento en ROS. Además, si la calidad de las imágenes es grande se pueden producir retrasos de computación por el gran coste de su procesamiento, lo que influirá negativamente en la localización del robot y en el mapa del entorno.

Otro factor importante a tener en cuenta en la aplicación de SLAM en exteriores es que las cámaras rgb no se comportan bien ante los cambios bruscos de luz y los resultados de SLAM se ven afectados enormemente por ello. Para mejorar el comportamiento del algoritmo ante estas situaciones se podría estudiar qué tipos de *features* convienen ser detectados en situaciones como estas y ver cómo afecta esto a los resultados.

Respecto a los experimentos realizados en interiores, se obtienen resultados bastante satisfactorios ya que los errores en la localización del robot en tres dimensiones obtenida por el algoritmo SPTAM SLAM tienen un error medio de centímetros respecto a la localización del *ground truth*.

Se ha demostrado que el despegue del UAV produce generalmente un efecto negativo en la estimación de su posición y que los resultados mejoran si el SLAM empieza a funcionar una vez que el robot ha despegado. También se ha demostrado que la estrategia de dar una vuelta de 360 grados en torno a z después del despegue para partir de un mapa inicial en el que basar la localización y así intentar reducir errores no es la mejor de las estrategias, por lo que habrá que buscar otras alternativas.

Por lo tanto, como futuras tareas quedan pendientes el estudio de la influencia de la detección de distintos tipos de *features* en entornos en exteriores donde la iluminación pueda afectar. En interiores es necesario la realización de pruebas donde el problema de mapeado y localización se realice una vez se haya realizado el despegue del UAV y descartar la estrategia de realizar un giro de 360 grados respecto del eje z para intentar partir de un mapa inicial, ya que realmente no es necesario y empeora los resultados de la estimación de la posición del robot.

Capítulo 6

Conclusión y desarrollos futuros

6.1. Conclusión

En la última década los UAV han mejorado su autonomía tanto energéticamente como computacionalmente lo que ha permitido dotar a estos robots de capacidades sensoriales y de procesamiento, produciendo un gran avance en tareas de seguimiento y localización.

Para que un UAV opere de forma autónoma se requiere de una buena estimación de su posición en tiempo real respecto del entorno en el que se mueve. En múltiples ocasiones ese entorno es desconocido por lo que se hace necesario la aplicación de técnicas de mapeado que proporcionen información al robot para poder localizarse. Los problemas de mapeado y localización están muy relacionados e incluso son dependientes el uno del otro. Es por ello por lo que surgen las técnicas de SLAM (*Simultaneous Localization and Mapping*).

Las técnicas SLAM han solido aplicarse para casos en que el vehículo es terrestre y principalmente en espacios en interiores. Son pocos los estudios realizados con vehículos aéreos y menos aún en casos en los que el robot se encuentre sumergido bajo agua, tema que está despertando bastante interés. En cuanto a sensores, actualmente el problema de SLAM puede considerarse resuelto cuando la información es proporcionada por dispositivos láser o sonar. Sin embargo, el SLAM visual es todavía una gran rama de investigación ya que tanto los sensores visuales como las técnicas de visión por computación son susceptibles de ser mejorados aún.

El objetivo de este proyecto es la generación de *datasets* multi-sensor del vuelo de UAVs que permitan el desarrollo de técnicas de mapeado y localización de estos vehículos sin la necesidad de GPS. Para ello se ha utilizado un láser 3D, una cámara estéreo, sensores de rango y una unidad de medición inercial. Una vez estudiadas sus características técnicas y especificaciones se ha podido concluir que los sensores de que se dispone aportan grandes prestaciones, consiguiendo sets de datos de bastante precisión y alcance por lo que se exigirá que los resultados del mapeado y localización obtenidos también sean bastante precisos.

De los experimentos en exteriores se han generado cuatro *bags* que han permitido la posterior aplicación de técnicas RGB-D SLAM. Se han obtenido tanto la nube de puntos del mapa, denso, como la estimación de la posición en dicho mapa. Sin embargo, no se ha podido conseguir el mapa completo del entorno por la iluminación. Según los resultados se ha podido concluir que los cambios bruscos de iluminación afectan al sensor RGB-D, produciéndose efectos negativos en el mapa y la localización calculados por el SLAM.

Respecto a los experimentos realizados en interiores, se ha tenido que trabajar con otro *hardware* distinto, marcado por el concurso EuRoC, pero de la misma naturaleza que los explicados en este proyecto, por lo que no ha supuesto ningún problema. Se han generado varios *bags* de datos del vuelo de un UAV y se han aplicado técnicas de *Stereo Parallel Tracking and Mapping* SLAM con el objetivo de cumplir con la tarea de localizar al vehículo. Los resultados obtenidos han sido comparados con el *ground truth*, proporcionado por cámaras VICON. Se ha justificado que dichos resultados son bastante satisfactorios ya que la estimación de la posición del robot mejora la realizada por la odometría visual, obteniéndose errores de algunas decenas de centímetros.

Definitivamente, el objetivo de este proyecto se ha cumplido e incluso se han podido aplicar técnicas de mapeado y localización obteniéndose resultados satisfactorios sin el uso de GPS. A continuación se van a proponer mejoras e investigaciones futuras acerca de este tema.

6.2. Desarrollos futuros

Las técnicas de SLAM visual todavía siguen siendo una amplia rama de investigación ya que tanto el desarrollo de los dispositivos visuales como el de las técnicas de procesamiento de imágenes están permitiendo obtener resultados que mejoran con creces los obtenidos con los sensores láser o sonar.

En este proyecto se han utilizado dos técnicas de SLAM. La primera de ellas ha sido un SLAM de tipo RGB-D, con el que se han obtenido resultados coherentes pero sólo en ciertas partes del mapa donde la iluminación no afecta al dispositivo. Uno de los estudios que podrían hacerse sería la influencia de la naturaleza de los *features* y detectores utilizados para realizar el mapa y obtener una buena localización respecto a él cuando se producen cambios de luz. Con las conclusiones obtenidas se podría conseguir evitar los efectos negativos que producen esos cambios sobre los resultados.

La otra técnica de SLAM aplicada ha sido de tipo SPTAM. Los resultados indican que la localización obtenida del robot es bastante buena. Sin embargo, se ha demostrado que los datos del despegue del UAV afectan negativamente a los resultados por lo que, a pesar de que se ha demostrado que sin ese despegue se reducen los errores, quedaría pendiente la realización de pruebas reales.

Una de las estrategias en las que se pensó a la hora de hacer pruebas fue la de realizar un giro de 360 grados en torno a z una vez que el UAV hubiera despegado para obtener un mapa del entorno antes de que el robot empezara a desplazarse. Lo que se pretendía con esto era reducir errores en la localización ya que inicialmente se tendría más información acerca del mapa. Como se ha concluido anteriormente, esta estrategia no produce mejoras en los resultados si no todo lo contrario. Por lo tanto, queda pendiente la exploración de otras estrategias que puedan reducir los errores aún más a los ya obtenidos.

A parte de los estudios que se proponen hacer para mejorar los resultados de estas dos técnicas de SLAM, también queda pendiente como trabajo futuro la realización de un vuelo en el que también se implemente el velodyne abordo del UAV. Con ello, se tendrán *bags* de datos con medidas de sensores de distinta naturaleza pero que tendrán en común el escenario y la trayectoria seguida (ya que todos están abordo del mismo robot). Se podrían aplicar técnicas de SLAM que se ajustara a cada uno de los sensores y con la estimación del mapa y la trayectoria obtenida de cada una de ellas, aplicar algoritmos de superposición. El objetivo sería obtener un mapa rico en información así como una estimación de la localización del robot con precisiones muy por debajo

de las obtenidas con la aplicación de las técnicas de SLAM por separado.

Bibliografía

[1] L. Merino, F. Caballero, J. R. M. de Dios, and A. Ollero, "Cooperative fire detection using unmanned aerial vehicles," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 1884–1889, April 2005.

[2] F. Ruggiero, M. A. Trujillo, R. Cano, H. Ascorbe, A. Viguria, C. Peréz, V. Lippiello, A. Ollero, and B. Siciliano, "A multilayer control for multirotor uavs equipped with a servo robot arm," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4014–4020, May 2015.

[3] S. Thrun, "Exploring artificial intelligence in the new millennium," *Robotic mapping: a survey*, pp. 1–36, 2003.

[4] AEROARMS project. Aerial Robotics System integrating multiple ARMS, "`www.aeroarms-project.eu/`," 30 de Junio de 2016.

[5] Aníbal Ollero Baturone, página personal, "`grvc.us.es/head-of-the-group/`," 30 de Junio de 2016.

[6] ARCAS project. Aerial Robotics Cooperative Assembly System, "`www.arcas-project.eu/`," 30 de Junio de 2016.

[7] A. Torres-González, J. R. M. d. Dios, and A. Ollero, "Efficient robot-sensor network distributed seif range-only slam," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1319–1326, May 2014.

[8] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[9] W. Kabsch, "A discussion of the solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 34, no. 5, pp. 827–828, 1978.

[10] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[11] J. Aulinas, Y. R. Petillot, J. Salvi, and X. Lladó, "The slam problem: a survey.," in *CCIA*, pp. 363–371, Citeseer, 2008.

[12] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[13] A. J. Davison and D. W. Murray, "Simultaneous localization and map-building using active vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 865–880, Jul 2002.

[14] C. Estrada, J. Neira, and J. D. Tardos, "Hierarchical slam: Real-time accurate mapping of large environments," *IEEE Transactions on Robotics*, vol. 21, pp. 588–596, Aug 2005.

[15] J. Leonard and P. Newman, "Consistent, convergent, and constant-time slam," in *IJCAI*, pp. 1143–1150, 2003.

[16] P. Newman, "On the structure and solution of the simultaneous localisation and map building problem," *Doctoral diss., University of Sydney*, vol. 41, 1999.

[17] J.-H. Kim and S. Sukkarieh, "Airborne simultaneous localisation and map building," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1, pp. 406–411, IEEE, 2003.

[18] J. Kim and S. Sukkarieh, "Autonomous airborne navigation in unknown terrain environments," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, no. 3, pp. 1031–1045, 2004.

[19] R. M. Eustice, H. Singh, J. J. Leonard, and M. R. Walter, "Visually mapping the rms titanic: Conservative covariance estimates for slam information filters," *The international journal of robotics research*, vol. 25, no. 12, pp. 1223–1242, 2006.

[20] Y. R. Petillot, J. Salvi, and E. Batlle, "3d large-scale seabed reconstruction for uuv simultaneous localization and mapping," *IFAC Proceedings Volumes*, vol. 41, no. 1, pp. 19–24, 2008.

[21] J. M. Sáez, A. Hogue, F. Escolano, and M. Jenkin, "Underwater 3d slam through entropy minimization," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 3562–3567, IEEE, 2006.

[22] W. Burgard, D. Fox, H. Jans, C. Matenar, and S. Thrun, "Sonar-based mapping with mobile robots using em," in *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pp. 67–76, MORGAN KAUFMANN PUBLISHERS, INC., 1999.

[23] L. M. Paz, P. Piniés, J. D. Tardós, and J. Neira, "Large-scale 6-dof slam with stereo-in-hand," *IEEE transactions on robotics*, vol. 24, no. 5, pp. 946–957, 2008.

[24] L. A. Clemente, A. J. Davison, I. D. Reid, J. Neira, and J. D. Tardós, "Mapping large loops with a single hand-held camera.," in *Robotics: Science and Systems*, vol. 2, p. 2, 2007.

[25] C. F. Olson, L. H. Matthies, M. Schoppers, and M. W. Maimone, "Rover navigation using stereo ego-motion," *Robotics and Autonomous Systems*, vol. 43, no. 4, pp. 215–229, 2003.

[26] M. Kaess and F. Dellaert, "Probabilistic structure matching for visual slam with a multi-camera rig," *Computer Vision and Image Understanding*, vol. 114, no. 2, pp. 286–296, 2010.

[27] G. Carrera, A. Angeli, and A. J. Davison, "Slam-based automatic extrinsic calibration of a multi-camera rig," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2652–2659, IEEE, 2011.

[28] D. Scaramuzza and R. Siegwart, "Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles," *IEEE transactions on robotics*, vol. 24, no. 5, pp. 1015–1026, 2008.

[29] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *International Symposium on Robotics Research (ISRR)*, vol. 2, 2011.

[30] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2014.

[31] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[32] G. Bradski and A. Kaehler, "Learning opencv, september 2008."

[33] S. Srinivasa, D. I. Ferguson, M. Vande Weghe, R. Diankov, D. Berenson, C. Helfrich, and H. Strasdat, "The robotic busboy: Steps towards developing a mobile robotic home assistant," 2008.

[34] F. Caballero, L. Merino, J. Ferruz, and A. Ollero, "Vision-based odometry and slam for medium and high altitude flying uavs," in *Unmanned Aircraft Systems*, pp. 137–161, Springer, 2008.

[35] C. Estrada, J. Neira, and J. D. Tardós, "Hierarchical slam: Real-time accurate mapping of large environments," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 588–596, 2005.

[36] P. Jensfelt, D. Kragic, J. Folkesson, and M. Bjorkman, "A framework for vision based bearing only 3d slam," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1944–1950, IEEE, 2006.

[37] S. Se, D. Lowe, and J. Little, "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks," *The international Journal of robotics Research*, vol. 21, no. 8, pp. 735–758, 2002.

[38] F. Moosmann and C. Stiller, "Velodyne slam," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pp. 393–398, June 2011.

[39] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 573–580, IEEE, 2012.

[40] Velodyne LiDAR, "`velodynelidar.com/`," 30 de Junio de 2016.

[41] STEREO LABS, "`www.stereolabs.com`," 30 de Junio de 2016.

[42] Xsens, "`www.xsens.com`," 30 de Junio de 2016.

[43] nanotron TECHNOLOGIES, "`nanotron.com/EN/index.php`," 30 de Junio de 2016.

[44] GitHub, stereolabs zed ros wrapper, "`github.com/stereolabs/zed-ros-wrapper`," 30 de Junio de 2016.

[45] Francis Colas, ROS Driver for Xsens MT, MTi, MTiG devices, "`www.ros.org/browse/details.php?distro=indigoname=xsens`$_{d}river$," $30deJuniode2016$.

[46] Grupo de Robótica, Visión y Control del Departamento de Ingeniería de Sistemas y Automática de la Unversidad de Sevilla, "grvc.us.es," 6 de Julio de 2016.

[47] European Robotics Challenges, "www.euroc-project.eu," 6 de Julio de 2016.

[48] Centro Avanzado de Tecnologías Aerospaciales, "www.catec.aero/administraci-n/historia-misi-n-y-vis 6 de Julio de 2016.

[49] Felix Endres, Juergen Hess, Nikolas Engelhard, "github.com/felixendres/rgbdslam$_v$2," $6deJuliode2016$.

[50] Fundación Andaluza para el Desarrollo Aerospacial, "www.catec.aero/administraci-n/fada.htm," 6 de Julio de 2016.

[51] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart, "A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 431–437, IEEE, 2014.

[52] Patrick Mihelich, Kurt Konolige, Jeremy Leibs, "www.asl.ethz.ch/," 6 de Julio de 2016.

[53] Patrick Mihelich, Kurt Konolige, Jeremy Leibs, "wiki.ros.org/stereo$_i mage_p roc$," $6deJuliode2016$.

[54] , "www.vicon.com/," 6 de Julio de 2016.

[55] T. Pire, T. Fischer, J. Civera, P. De Cristóforis, and J. Jacobo berlles, "Stereo Parallel Tracking and Mapping for robot localization," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS) (Accepted)*, 2015.